TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Realitat Augmentada i Geolocalització amb iPhone

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Xavier Lagunas Calpe

DIRECTOR: Juan López Rubio

DATA: 20 de Maig de 2011

Títol: Realitat Augmentada i Geolocalització amb iPhone

Autor: Xavier Lagunas Calpe

Director: Juan López Rubio

Data: 20 de Maig de 2011

Resum

La irrupció del telèfon mòbil *iPhone* a l'any 2007 va revolucionar el mercat de les telecomunicacions, concretament el sector mòbil. En poc més de tres anys, aquest sector ha patit una transició dràstica: els terminals mòbils –en especial els *smartphones* o telèfons intel·ligents- han passat de ser emprats per poc més que trucar o intercanviar missatges de text a ser dispositius que aglutinen gairebé totes les funcions que un ordinador de sobretaula pot dur a terme al palmell de la mà.

D'altra banda, la tendència actual és que aquests telèfons intel·ligents vinguin equipats amb sistemes de posicionament tals com GPS o brúixoles digitals que conjuntament amb els dispositius de reproducció i enregistrament de continguts multimèdia obren un ventall de possibilitats enorme en el món de les tecnologies 2.0.

Aquest projecte es centrarà en l'estudi de *l'iOS*, el sistema operatiu d'*Apple* destinat als dispositius mòbils, i el desenvolupament d'una aplicació per un dels seus dispositius, l'*iPhone*.

La aplicació, VidGeoLocator, té com a funcionalitat la publicació i reproducció en *streaming* de contingut audiovisual –vídeo- directament des del telèfon mòbil. A més, gràcies a les tècniques de *geolocalització*, l'usuari podrà veure, ja sigui sobre un mapa o mitjançant realitat augmentada, el lloc on s'han enregistrat els vídeos.

Title: Augmented Reality and GeoLocation with iPhone

Author: Xavier Lagunas Calpe

Director: Juan López Rubio

Date: May 20th, 2011

Overview

iPhone's landing in 2007 torn upside down the telecommunications market, particularly the mobile sector. In just over three years, this sector has suffered a dramatic transition: mobile devices, particularly smartphones, have evolved from being devices used just for calling or sending text messages to become pocket-sized powerful machines which can perform almost as a desktop computer would.

Nowadays, smartphones are equipped with positioning systems such as GPS or digital compass which, in conjunction with a mix of playback and recording multimedia capabilities, offer to their owners a whole new era of possibilities in the world of 2.0 technologies.

This project will focus on the study of the IOS, the *Apple* operating system intended for mobile devices, with one main goal: to develop an application for his flagship device, the *iPhone*.

The application, VidGeoLocator, has the functionality to publish and view multimedia content by streaming video directly to the mobile phone. Moreover, thanks to geo-location techniques, the user will have the chance to choose between two different views to locate where were the videos recorded: over a map view or through an augmented reality layer.

A Daniel Rodríguez i Juan López: per ajudar a fer d'aquest projecte una realitat.

A Antonio Santoyo: per la seva comprensió i flexibilitat.

A Carlos R., Raquel L. i tots els companys d'i2cat: per la seva insistència i ajuda.

A la meva família: per tindre la paciència necessària per dur-me fins aquí i fer-me costat sempre.

ÍNDEX

ÍNDEX DE TAULES

ÍNDEX DE FIGURES

INTRODUCCIÓ

Des de l'aparició dels telèfons intel·ligents i posteriorment dels *tablets*, el sector de les TIC ha patit un canvi molt ràpid i brusc. Els primers fabricants que han apostat fort per adaptar-se a les bondats d'aquests sistemes han obert un nou mercat i ja hi ha qui s'aventura a dir que s'està iniciant una nova etapa *post-pc*.

Agradi o no, hi ha una realitat ben palpable i és que el factor de la mobilitat està guanyant cada cop més adeptes i, a diferencia del que passava en temps anteriors, aquest factor ja no està renyit amb la potència dels dispositius.

Ans al contrari, el fet de poder-se emportar a qualsevol lloc dispositius que gairebé igualen en potencia i prestacions als ordinadors de sobretaula, ha obert un nou ventall de possibilitats que a dia d'avui són ja una realitat. Termes com *geolocalització* o realitat augmentada no són desconeguts al públic general i això indica que aquestes tecnologies són ja, en aquests moments, de domini públic.

En aquest context s'emmarca l'estudi d'aquest projecte. Per realitzar-lo, s'ha triat una de les primeres plataformes que va apostar fort per aquest nou mercat en primera instancia en forma de telèfon intel·ligent –l'*iPhone*- i posteriorment en forma de *tablet* -l'*iPad*-: *Apple*.

La meta final d'aquest projecte consisteix en el desenvolupament d'una aplicació per *iPhone* que faci ús de les tecnologies de mobilitat disponibles al terminal. En concret, es proposa realitzar una aplicació basada en la compartició, reproducció en *streaming* i enregistrament de continguts multimèdia –vídeos- geolocalitzats entre usuaris amb dispositius *iPhone*.

Per dur a terme tal empresa caldrà realitzar en primera instancia un estudi del sistema i les tecnologies que aquest sistema implementa per estudiar la viabilitat del projecte.

L'estructura d'aquest document es divideix en 5 capítols, que aniran enfilant els diferents objectius que faran possible la consecució de dita meta.

Els objectius que s'intentaran cobrir són:

- Estudi de l'ecosistema mòbil d'*Apple*.
- Estudi del llenguatge de desenvolupament per *iOS*.
- Estudi de tecnologies incloses al dispositiu *iPhone*.
- Estudi sobre la realitat augmentada.
- Disseny d'una aplicació pel sistema operatiu *iOS*.
- Implementació d'una aplicació per *iPhone*.

CAPÍTOL 1. iOS: EL SISTEMA OPERATIU DE L'iPhone

1.1. INTRODUCCIÓ

L'iOS és el sistema operatiu amb que actualment funcionen tots els dispositius mòbils de l'ecosistema d' *Apple*.

La meta d'aquest projecte de final de carrera es desenvolupar una aplicació per *iPhone* i per assolir-la és cabdal conèixer en profunditat tant el nucli del sistema que el gestiona com les capacitats i funcionament del propi terminal. És per això que aquest primer capítol s'endinsarà en el propi *iOS*, concretament l'iOS4.

En aquest capítol s'analitzarà primerament el procés evolutiu de la pròpia plataforma i posteriorment es donarà una mirada molt més tècnica a les possibilitats del propi sistema operatiu des del punt de vista del programador; des del llenguatge de programació i l'entorn de desenvolupament fins al procés de distribució d'una aplicació.

Finalment i degut a situació actual cada cop més polaritzada del mercat dels *smartphones* es proposarà tant un estudi d'avantatges e inconvenients d'un sistema tancat, com és el propi *iOS*, com una comparativa entre els dos models més oposats: l' obert *d'Android* i el tancat *d'iOS*.

1.1.1. Història i Evolució de la plataforma

Apple va desvetllar l'existència de l' *iPhone* a una de les seves conferències el 9 de gener de 2007. Com que no van anunciar en cap moment quin sistema operatiu el governava, tothom va donar per fet que es tractava del mateix OS X que utilitzen els ordinadors de la pròpia *Apple*. No va ser fins un any més tard, amb la introducció de la primera versió de *l'SDK*, concretament al 6 de març del 2008, que van decidir batejar-lo com a *iPhone OS*.

Posteriorment, a la conferencia anual sobre els dispositius iPod (al setembre d'aquell mateix any), es produeix el llançament d'un segon terminal mòbil igual a l'*iPhone* en aparença i capacitats multimèdia però sense la possibilitat de realitzar trucades: l' iPod Touch.

Aquest fet, va resultar determinant, ja que apropava el sistema a un mercat molt menys poderós -econòmicament parlant- i expandia dràsticament el nombre de terminals basats en *iPhone* OS per tot el món. És en aquest moment que els programadors comencen a veure el potencial d'usuaris que hi ha en aquesta plataforma.

Amb l'aparició del primer SDK per l'*iPhone*, apareix també la *App Store*, una tenda virtual on els usuaris dels terminals basats en *iPhone OS* es descarreguen les aplicacions desenvolupades no només de la pròpia companyia sinó també de tercers. El model de negoci esdevé un èxit absolut.

Dos anys més tard, el 27 de Gener de 2010, Steve Jobs, presenta en societat l'*iPad*, el tercer dels dispositius que funcionarien sota l'*iPhone* OS.

Posteriorment, a la conferència anual de l'*iPhone*, durant la presentació del següent *iPhone* –l' *iPhone* 4- Jobs anuncia que el sistema operatiu, degut a la fragmentació dels terminals que el suporten ja no es dirà *iPhone* OS, sinó que passa a anomenar-se iOS i presenta la nova versió, *l'iOS* 4.

Actualment, s'ha afegit, per una banda, l'*Apple* TV –a grans trets, un proveïdor de televisió a la carta en *streaming* i videoclub online- i per l'altre la segona versió de l'*iPad*, l'*iPad* 2.

1.2. CARACTERÍSTIQUES DEL SISTEMA OPERATIU

Abans de començar a tractar aquest apartat cal fer una consideració, l'apartat anterior conclou amb la irrupció de la nova versió de l' *iOS*, la versió 4. Aquesta versió compta amb importants millores però en contrapartida no és vàlida per a tots els terminals degut als requisits hardware.

Per a la realització del treball s'ha utilitzat el darrer SDK, la versió 4, així que les característiques que es citen a continuació són d'aquesta versió.

Un cop aclarit aquest apartat, i d'acord amb la Fig. 1.1, s'introduiran les capes que conformen l'arquitectura de l' *iOS*:

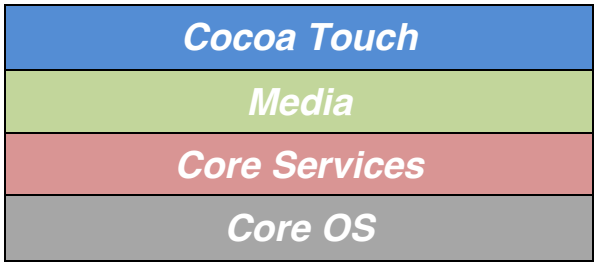

Fig. 1.1 Capes que composen *l'iOS*

iOS és un sistema *UNIX* basat en una versió modificada del *kernel* de Mac OS X. Aquest *kernel* està basat en Darwin que és un sistema operatiu *open source POSIX (Portable Operating System Interface for uniX).*

Tot i ser un sistema operatiu UNIX, no es pot accedir al hardware del sistema directament sinó que únicament es permet accedir als dispositius del terminal mitjançant els serveis que proporcionen les capes que es mostren a la figura 1.1.

Com sol ser habitual, les capes inferiors doten de la possibilitat de programar a molt baix nivell, mantenint un control total sobre el comportament del codi, mentre que, contràriament, les capes superiors abstrauen al desenvolupador d'aquests grau d'aprofundiment en el codi, dotant de mètodes i crides

orientades a objecte de molt alt nivell i agilitzant molt més el procés de desenvolupament d'una aplicació.

1.2.1. La capa Cocoa Touch

La capa *Cocoa Touch* conté tots els *frameworks* essencials per crear aplicacions per *l'iOS*. Aquesta capa defineix la infraestructura bàsica d'una aplicació i dóna suport a tot el conjunt de *frameworks* i tecnologies disponibles a la figura 1.2.

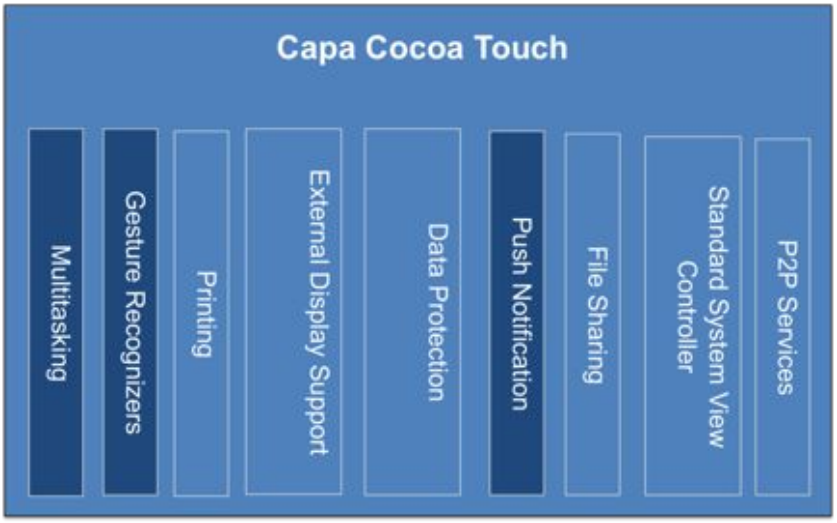

Fig. 1.2 *Frameworks* pertanyents a la capa *Cocoa Touch*

En aquesta mateixa figura s'observen ressaltats els 3 aspectes més rellevants i que seran explicats a continuació.

El *Multitasking* és una de les tecnologies estrella dins l' iOS4. Fins a aquesta última versió, no hi havia cap forma en que dues o més aplicacions poguessin estar executant-se alhora. Aquest va ser un dels principals motius que va dur a crear les anomenades notificacions *Push* en versions anteriors del sistema operatiu que veurem més endavant.

En les versions anteriors del sistema operatiu, cada cop que, trobant-se dins d'una aplicació, es polsava el botó de *HOME* del terminal, es forçava la sortida de qualsevol aplicació. A la darrera versió, per contra, un cop pressionat el botó per tornar al menú principal, l'aplicació –si ho implementa- s'envia a segon pla, en un estat de repòs.

Tot seguit, i en la gran majoria de casos, un cop una aplicació ha passat a estar en repòs deriva cap a un estat de suspensió (està carregada en memòria però no s' executa cap línia de codi).

Tot i anomenar-se *multitasking*, no és estrictament cert que *l'iOS* pugui executar dues o més aplicacions alhora, ja que es limita a congelar l'estat de la aplicació que passa a segon pla, deixant-la en repòs fins que es torni a cridar. Tot i això, es tan ràpida la transició de l'estat de repòs a actiu d'una aplicació que sembla certament que la aplicació no hagués estat mai aturada.

Ja que s'ha començat parlant del *multitasking*, a continuació es tractarà una altra funció força relacionada: l' *Apple Push Notification Service*. Aquest servei, era la solució que els enginyers d'*Apple* havien trobat a les versions prèvies a *l'iOS4* per solucionar el problema de la multitasca.

Consideri's el següent exemple: Un usuari esta esperant l'arribada d'un correu amb les dades d'una entrevista per feina. Obre l'aplicació de correu del seu terminal i comprova que per desgracia encara no ha rebut el missatge. En aquell moment s'adona que té una visita mèdica aquell mateix dia però no recorda a quina hora exactament era, així que obre l'aplicació del calendari per comprovar-ho. Passen les hores i el correu no arriba. Per la nit, quan l'usuari arriba a casa abatut per la oportunitat perduda, engega el seu ordinador i al consultar el seu gestor de correu comprova que aquell correu tan important ha arribat fa 6 hores i el seu telèfon intel·ligent no l'ha notificat. Gràcies a això l'usuari ha perdut una entrevista de treball, i tot degut a que el seu *smartphone* no té multitasca.

Aquest exemple il·lustra el problema de la multitasca en les primeres versions del terminal. El fet de sortir, de canviar d'aplicació o simplement de tornar al menú principal tancava la aplicació que estigués en funcionament en aquell moment. Per tant, totes les aplicacions que requerissin de sincronització amb un servidor extern perdien la comunicació.

Per solucionar aquest problema, com ja s'ha dit, *Apple* crea el *Push Notification Service*, que no és res més que un servei -no una aplicació- que esta sempre en funcionament en segon pla pendent de capturar totes les notificacions que arriben des dels servidors *d'Apple*.

És un servei centralitzat a cada terminal, per tant, totes les notificacions, independentment de la aplicació a la que vagin destinades, aniran a parar al mateix servei que les anirà posant en una cua en ordre d'arribada (cua *FIFO*) .

D'aquesta manera, encara que la aplicació no estigui activa, quan es generi un esdeveniment a un servidor extern, aquest servidor extern comunicarà la alerta al servidor *d'Apple* i aquest generarà una notificació que alertarà sobre la naturalesa i l'aplicació a la que va destinada a l'usuari final.

Finalment, la última secció que cobrirem dins de la capa *Cocoa Touch* és amb tota seguretat la tecnologia més emprada en la manipulació del terminal: El reconeixement de gestos tàctils.

La característica més destacada de l'*iPhone* respecte als seus competidors des del primer dia de la seva presentació ha estat sense dubte la capacitat multi tàctil de la seva pantalla. Aquesta capa ens dota del que anomenen reconeixement gestual que no és més que un conjunt de patrons per als gestos tàctils. Gràcies a aquests patrons es pot distingir (sense haver d'afegir cap línia de codi a la aplicació) entre pulsació, pessic (per fer zoom), arrossegament, rotació o pulsacions llargues. Per suposat també permet la possibilitat de crear i registrar els gestos propis si així s'estima necessari.

1.2.2. La capa *Media*

Fig. 1.3 *Frameworks* pertanyents a la capa *Media*

A la capa *media*, es troben les tecnologies que gestionen totes les aplicacions que tracten amb tecnologies de gràfics, vídeo o àudio.

Pel que fa a les tecnologies gràfiques, hi ha 6 diferents *frameworks* depenent de quin tipus de gràfic es gestioni a la aplicació (No s'utilitzarà el mateix framework per renderitzar un objecte en 3d que per mostrar una foto capturada per la càmera, per exemple).

Taula 1.1 Comparativa de tecnologies gràfiques presents a l'iOS

	Vectors Bidimensionals	Imatges	Animacions	Gràfics 2D	Gràfics 3d	Acceleració Hardware	Llibreria fotos del telèfon
Quartz Core Graphics	✓	✓	✗	✓	✗	✗	✗
Core Animation	✓	✓	✓	✗	✗	✗	✗
OpenGL ES	✗	✗	✓	✓	✓	✓	✗
Core Text	✗	✗	✗	✓	✗	✗	✗
Image I/O	✗	✓	✗	✗	✗	✗	✓
Assets Library	✗	✗	✗	✗	✗	✗	✓

La taula 1.1, sobre aquestes línies mostra una comparativa entre les tecnologies que gestionen els gràfics a *l'iOS* i la funció que desenvolupen cadascuna d'elles.

- *Core Graphics* o *Quartz*: s'encarrega de renderitzar gràfics basats tant en vectors bidimensionals com en imatges.

- *Core Animation*: Complementa a *Core Graphics* ja que s'encarrega de gestionar les animacions de les imatges renderitzades per *Core Graphics*

- *OpenGL ES*: Proporciona suport per renderitzar en 2D o en 3D que requereixen d'acceleració per hardware (com els videojocs).

- *Core Text*: Proporciona suport per renderitzar fonts de text

- *Image I/O*: Proporciona mètodes per llegir i escriure en formats d'imatge

- *Assets Library*: Dóna accés a la llibreria de fotos i vídeos de l'usuari del telèfon.

Pel que fa a les tecnologies d'àudio, a l'*iOS* es proporcionen diferents formes (tal i com passa amb els gràfics) de reproduir–ne i enregistrar-ne. A la taula 1.2 es troba la llista dels còdecs suportats

Taula 1.2 Còdecs d'audio disponibles a *iOS*

Tecnologies d'audio Suportades
AAC
Apple Lossless Audio ALAC
Llei A
Llei µ
IMA/ADPCM
PCM
Intel IMA ADPCM
Microsoft GSM 6.10
AES3-2003

A continuació es repassaran ordenadament de més a menys alt nivell els *frameworks* per gestionar l'àudio:

- *Media Player framework*: Proporciona un accés simple a la llibreria de l'*iTunes* de l'usuari, donant suport per reproduir cançons i llistes de cançons.

- *AV Foundation framework*; Proporciona un conjunt d'instruccions en *Objective*-C per gestionar tant la reproducció com la enregistrament d'àudio.

- *Core Audio framework*: És el *framework* de més baix nivell i dóna accés al *buffering* i a la reproducció de contingut d'àudio multicanal, ja sigui local o en *streaming*.

Finalment, només resta parlar de les tecnologies de vídeo (Taula 1.3). Tal com s'ha fet amb l'àudio, els còdecs suportats en vídeo són H.264 i MPEG-4.

Taula 1.3 Còdecs de vídeo disponibles a *iOS*

	Enregistrar H.264	Reproduir H.264	Enregistrar MPEG-4	Reproduir MPEG-4
UIImagePicker	✓	✗	✓	✗
Media Player	✗	✓	✗	✓
AVFoundation	✓	✓	✓	✓
Core Media	✓	✓	✓	✓

Pel que fa als *frameworks* que gestionen el vídeo es troben:

- *UIImagePickerController*: és la interfície Standard mitjançant la qual s'enregistra vídeo en dispositius amb una càmera que estigui suportada.
- Media Player *Framework*: *Framework* que permet de forma molt simple reproduir vídeos.
- *AVFoundation framework*: Aquest *framework* és compartit amb l'àudio. En aquest cas, proveeix de crides en llenguatge *Objective*-C ja sigui per enregistrar o per reproduir vídeos al terminal.
- *Core* Media: S'hi troben definides les estructures de dades de baix nivell utilitzades pels *frameworks* de més alt nivell i a més a més proveeix d'interfícies de baix nivell per treball directament per a aquestes estructures

1.2.3. La capa Core Services

Aquesta capa conte els serveis del sistema més fonamentals i que integren totes les aplicacions. Encara que quan es desenvolupi una aplicació no s'utilitzin directament les tecnologies i serveis d' aquesta capa, de ben segur que l'aplicació en farà ús, ja que una gran quantitat de *frameworks* estan construïts al damunt d'aquesta capa.

A la Taula 1.4 es poden observar 4 de les tecnologies més rellevants d'aquesta capa:

Taula 1.4 Tecnologíes clau a la capa Core Services

Tecnologies capa *Core Services*
In-App Purchase
SQLite
XML Support
Grand Central Dispatch

In-App Purchase és la tecnologia que dota al desenvolupador de la possibilitat d' oferir serveis extra a l'usuari final (vendre) des de l'interior de l' aplicació.

La llibreria *SQLite* dóna la possibilitat d' utilitzar una base de dades incrustada dintre de la aplicació. Aquesta solució evita la necessitat d' emmagatzemar la informació de la aplicació a bases de dades. Per contra, cal recordar que la memòria per aplicació de *l'iOS* és limitada i aquesta llibreria no serà recomanable per aplicacions que requereixin d'un ús molt intensiu de bases de dades relativament grans.

L'iOS també dóna suport a les tecnologies XML. Mitjançant el *framework NSXMLParser* es possibilita *parsejar* o escriure en fitxers XML sense la major dificultat.

En darrer lloc, *Grand Central Dispatch;* una tecnologia directament obtinguda del sistema operatiu OS X del ordinadors d'*Apple*. Aquesta tecnologia està enfocada a optimitzar el suport de les aplicacions per a processadors de diversos nuclis.

Taula 1.5 *Frameworks* de la capa *Core Services*

Frameworks Core Service
AddressBook Framework
CFNetwork Framework
Core Data Framework
Core Foundation Framework
Core Location Framework
Core Media Framework
Core Telephony Framework
Event Kit Framework
Foundation Framework
Mobile Core Services Framework
Quick Look Framework
Store Kit Framework
System Configuration Framework

Com s'ha començat dient en aquest apartat, aquesta capa disposa de tecnologies de molt baix nivell, com el *Grand Central Dispatch*, que difícilment són utilitzades per el programador directament. Per contra, és la capa més prolífica pel que fa a *frameworks*, com s'aprecia a la Taula 1.5, i, és precisament degut a que les tecnologies són de tan baix nivell i tan bàsiques que en deriven tants.

És absolutament imprescindible que els *frameworks* siguin ben optimitzats, no tindria sentit que una companyia limités al programador a utilitzar únicament les seves eines si aquestes no fossin bones i com és lògic els *frameworks* del sistema han de ser creats de la forma més eficient possible; això només

s'aconsegueix mitjançant la flexibilitat més gran que proporciona treballar al nivell més baix.

Per això, tot i ser de propòsits tan diversos, tots els *frameworks* bàsics provenen d'aquesta capa.

1.2.4. La capa *Core OS*

Per acabar amb el repàs de les capes que componen *l'iOS*, es troba la capa *Core OS*. En aquesta capa apareixen les característiques de més baix nivell que composen el sistema i que es troben a la Taula 1.6.

Taula 1.6 *Frameworks* de la capa *Core OS*

Accelerate Framework
External Accessory Framework
Security Framework
System

L'Accelerate Framework és una altra de les novetats de *l'iOS* 4.0. Són un conjunt de crides i funcions que ens permeten realitzar entre d'altres càlculs complexos (com per exemple els trobats per tractar processat digital del senyal) amb un gran avantatge sobre qualsevol codi propi que es pugui desenvolupar i és que està optimitzat per cadascun dels dispositius que funcionen amb *iOS*.

L'External Accessory framework, com el seu nom indica, s'encarrega de proporcionar les funcionalitats necessàries per establir una comunicació entre el terminal i qualsevol dispositiu que s'hi connecti (ja sigui via el Dock o via Bluetooth). Simplement s'encarrega, d'establir i gestionar un enllaç entre dispositius, la comunicació queda a mans del programador.

El *Security framework* s'encarrega de gestionar la seguretat de les dades de les aplicacions. Tot i trobar-se en una capa tan baixa no gestiona la seguretat del sistema, presumiblement per evitar que aquesta pugui ser compromesa, simplement s'encarrega de proporcionar mètodes per gestionar certificats, claus (ja siguin públiques o privades) i funcions de *HASH* com pugui ser el *MD5* (que serà utilitzat en aquest projecte).

El sistema com a tal, engloba l'entorn del *kernel*, divers i interfícies *UNIX* de baix nivell del sistema operatiu. El *kernel* es basa, com tots els sistemes operatius d'*Apple*, en el *Mach kernel* [1] i és responsable de tots i cadascun dels aspectes del sistema operatiu. Gestiona el sistema virtual de memòria, els *threads*, el sistema de fitxers, la xarxa i la comunicació entre processos. Els *drivers* fan d'interfície entre el hardware disponible i els *frameworks* del sistema.

Afegir que, per raons de seguretat, l'accés al *kernel* i als *drivers* està restringit a un conjunt limitat de *frameworks* del sistema.

1.3. OBJECTIVE-C: EL LLENGUATGE DE L'*IOS*

En aquest apartat s'introduirà el llenguatge que s'utilitza per desenvolupar pels terminals *d'iOS*.

Per fer-ho, s'introduirà breument el llenguatge: la seva provinença, la justificació per part de la empresa per utilitzar-lo com a llenguatge vehicular de tot el seu ecosistema, les influencies d'altres llenguatges i posteriorment es posaran exemples de la seva sintaxi.

1.3.1. Per què Objective-C?

Per a tot aquell programador que no s'hagi topat mai amb el desenvolupament d'una aplicació per l'entorn OS X –el sistema operatiu dels ordinadors d'*Apple*- o anteriors versions del mateix, probablement pensi que l' *Objective*-C és un llenguatge creat per a desenvolupar en *iOS* únicament; res més allunyat de la realitat.

El fet que aquest llenguatge sigui tan desconegut es déu a que a l'actualitat, l'únic ecosistema que en fa ús és el d'*Apple*. El software que gestiona des dels ordinadors de sobretaula fins al recent *iPad* 2, sense oblidar tota la gama *d'iPods* està escrit íntegrament en *Objective*-C.

Llavors, si té tants dispositius disponibles, per què no està tan estès? La resposta és fàcil, la portabilitat.

El gran inconvenient que presenta aquest llenguatge és que cap altra plataforma en fa ús, per tant, si es desenvolupa un programa en aquest llenguatge només funcionarà amb aquest reduït rang de dispositius i, si aquest mateix programa es volgués utilitzar sobre qualsevol altre sistema (per exemple Windows), s'hauria de reescriure de cap i de nou.

Aquest fet, lògicament, fa que donada la quota de mercat d'ambdues plataformes, sobretot fins fa ben poc on la desigualtat quedava molt més palesa, els desenvolupadors es decantessin pel sistema que tenia una extensió més gran i per tant un potencial de clients major.

Un cop aclarit aquest aspecte, la pregunta realment important és, per què es tria aquest llenguatge com a columna vertebral dels sistemes d'*Apple*?

L'any 1988 Steve Jobs, un dels co-fundadors d'*Apple Inc*, crea la empresa *NeXT Computers* (empresa dedicada a la fabricació de hardware i software) després de ser forçat a dimitir de la pròpia *Apple* degut a mals resultats financers per la seva mala gestió.

Aquell mateix any, *NeXT* llicència l' *Objective*-C de *StepStone* (el llenguatge en si ja existia i aquesta empresa n'era la propietària) i publica el seu propi

compilador *d'Objective*-C, les biblioteques en les quals es van basar la interfície d'usuari *NextStep* i el software per construir les pròpies interfícies.

Per fer-ho ras i curt, Apple compra *NeXT* a l'any 1996 i col·loca de nou a Steve Jobs al capdavant de la empresa.

A partir d'aquest moment, es comença a desenvolupar el software del Mac OS X de cap i de nou, escrit i basat íntegrament en les eines desenvolupades per la pròpia *NeXT*, passant a ser la pedra angular de tots els sistemes que es van desenvolupar a partir d'aquell moment fins arribar a l'actualitat.

Com a curiositat i com a mostra d'aquesta relació entre *Next Step* i *Objective*-C totes les classes primitives *d'Objective*-C (presents en el desenvolupament tant d'IOS com de OS X), com per exemple *NSInteger, NSFloat, NSString*,etc. tenen com a prefix NS que prové precisament de *Next Step*.

1.3.2. Descripció del llenguatge

El llenguatge *Objective*-C es defineix com un petit –però potent- conjunt d'extensions per al llenguatge estàndard ANSI C.

Les seves addicions a l'ANSI C es basen principalment en *Smalltalk*, un dels primers llenguatges de programació orientats a objectes. *Objective*-C està dissenyat per dotar a C de capacitats completes de programació orientada a objectes, i fer-ho d'una manera senzilla i directa.

Així doncs trobem que *Objective*-C beu de dues fonts, ANSI C i *SmallTalk*.

Pel que fa a ANSI C, és important dir que *Objective*-C actua com una capa de nivell superior que és completament compatible amb totes les funcions i requeriments d'ANSI C fins a tal punt que es perfectament possible compilar codi amb un compilador *d'Objective*-C que estigui escrit en *ANSI* C.

```
#include <stdio.h>

int main(int argc, char *argv[]){
        printf("Codi Ansi C pur");
        return 0
}
```

```
#import <stdio.h>

Int main(int argc, char* argv[]){
    printf("Codi Ansi C sobre Obj-C);
        return 0;
}
```

Com es veu a la comparativa superior, les dues crides són idèntiques en mètodes i funcions. Fan exactament el mateix codi i no es podria distingir quin codi correspon a quin llenguatge si no fos per les diferencies entre les directives #*include* (pròpies d' ANSI C) i #*include* (habituals a *Objective*-C).

Com a apunt, només indicar que *Objective*-C també suporta la directiva #*include* ja que sino totes les llibreries escrites en ANSI C o C++ que incorpora el llenguatge no serien compatibles.

La diferència entre les directives #*import* i #*include* és mínima. Des de la documentació d'*Apple* [2] s'indica que la diferencia és que la directiva #import inclou directament codi de protecció a la doble inclusió en detriment de la directiva #*include* on si es vol protegir una capçalera s'han d'afegir les directives de *preprocessat* #*ifndef* i #*endif*.

Tanmateix es segueix un conveni en pro de la portabilitat del codi en que totes les classes d'ANSI C o C++ que s'utilitzin dins d'una aplicació *d'Objective*-C portaran les directives #*include* pròpies del seu llenguatge, reservant només la directiva #import a les classes *d'Objective*-C estrictament.

Pel que fa a l'altre gran influencia d'*Objective*-C, *SmallTalk*, comentar que és un dels considerats primers llenguatges orientats a objectes, i que la seva filosofia principal és que tot és un objecte, des dels nombres reals passant pel propi entorn *Smalltalk*.

Les característiques d'aquest llenguatge són:

- Orientació a objectes pura
- Tipus Dinàmics
- Interacció entre objectes mitjançant l'enviament de missatges
- Herència simple i amb arrel comuna
- Reflexió computacional completa
- Recollida d'escombraries
- Compilació en temps d'execució

A la següent taula trobem una comparativa entre *Objective*-C, C++ i *SmallTalk*).

Taula 1.7 Comparativa entre llenguatges i POO

	SmallTalk	*Objective*-C	C++
Comprovació tipus	Dinàmic	Dinàmic o estàtic	Estàtic
Accés noms mètodes en *runtime*	✓	✓	×
Accés noms de classe en temps d'execució	✓	✓	×
Forwarding	✓	✓	×
Herència	Barreja	Simple	Múltiple
Classe Arrel	*Object*	*Object*	NO
Nom del receptor	*Self*	*self*	*this*
Dades privades	✓	✓	✓
Mètodes privats	✓	×	✓
Variables de Classe	✓	×	✓
Col·lector d'escombraries	✓	✓	×

Com es desprèn de la taula 1.7, *l'Objective*-C és força similar en estructura a *l'SmallTalk* mentre que difereix en força aspectes de C++.

Un dels primers canvis més significatius a primera vista quan es comença a inspeccionar aquest llenguatge és la sintaxi que utilitza:

```
[Objecte mètode:valor];
```

```
Objecte.mètode(valor)
```

Fig. 1.4 Comparativa sintaxi *Objective-C* i sintàxi clàssica

A la figura 1.4, tot i que les noves versions també la suporten, *Objective*-C canvia la sintaxi clàssica de llenguatges orientats a objecte -on és habitual referir-se al mètode d'un objecte mitjançant un punt- per una forma totalment diferent –heretada del llenguatge *SmallTalk*-; es canvia el concepte d'invocació de mètode d'un objecte tradicional pel d'enviament de missatges entre instàncies d'objectes.

A la següent figura (Fig. 1.5) observarem amb més deteniment el que a *Objective*-C s'anomena enviament de missatges:

Fig. 1.5 Format dels missatges a *Objective-C*

A priori sembla que sigui un simple convencionalisme i que en última instancia el que s'anomena missatge sigui simplement un mètode canviat de nom però realment hi ha una diferencia entre les dues possibilitats.

La diferència entre els conceptes radica en cóm s'executen els dos tipus de codi segons sigui un missatge o un mètode. En un llenguatge com C++ el nom del mètode queda associat a una secció del codi de la classe pel compilador, mentre que al llenguatge *SmallTalk* i per extensió a *Objetive-C*, el missatge segueix essent simplement un nom i serà resolt en temps d'execució. Això vol dir que l'objecte receptor del missatge serà qui s'encarregui –si pot- d'interpretar-lo.

Al fil d'aquesta diferència cal comentar que *Objective-C* també hereta *d'Smalltalk* la capacitat d'introspecció d'una classe. Això vol dir que es pot preguntar a una classe de quin tipus és i si respondrà a un determinat tipus de missatge en temps d'execució.

1.4. AVANTATGES E INCONVENIENTS D'UN SISTEMA TANCAT

Com a conclusió d'aquest capítol és adient comentar les bondats i inconvenients que es desprenen d'un sistema tancat com és *l'iOS*.

Pel que fa a l'experiència d'usuari, s'ha demostrat que és un bon sistema, ja que garanteix a tots els usuaris finals que no tindran problemes en adaptar-se a cap dispositiu nou de la marca.

Això no és una casualitat sinó que és fruit d'una estratègia molt astuta: No hi ha cap tipus de transició ni de corba d'aprenentatge per a un usuari que posseeix un *iPhone* i, posteriorment, adquireix un *iPad* ja que la interfície és exactament igual.

Per altra banda, el fet que tot el parc de dispositius *iOS* sigui limitat garanteix que tot el software que es desenvolupa per un dispositiu en concret –posem l'*iPhone* 4 que és el cas del projecte- funcionarà i es comportarà sobre tots els terminals de la mateixa manera ja que en software i en hardware tots els dispositius són exactament iguals i per tant tots estaran perfectament actualitzats (exceptuant els models que per antics no puguin suportar les noves versions del sistema, restant descatalogats).

Per contra en un sistema obert com per exemple *Android* -probablement l'amenaça més gran a l'actualitat de *l'iOS*- degut a la seva política diametralment oposada té molts problemes de fragmentació de mercat.

A nivell intern, no tots els terminals que utilitzen el sistema operatiu de *Google* que es llencen al mercat funcionen –o simplement es poden actualitzar- amb la última versió disponible, fet que força al programador a descartar una gran massa d'usuaris potencials de la seva aplicació en funció de la versió en que compili el seu programa. És tan crític aquest aspecte que en determinats moments han calgut notes de premsa per assegurar que un terminal suportaria actualitzacions

Prenent en compte l'experiència d'usuari, al contrari que *iOS*, *Android* ofereix la llibertat a la companyia que l'utilitzi de modificar la interfície del sistema. Per tant, tot i que internament dos dispositius siguin gairebé iguals com per els que veiem a la figura 1.6, un usuari inexpert veurà dos sistemes diferents i això repercuteix negativament ja que la corba d'aprenentatge o adaptació en aquest cas -encara que mínima- existeix.

Fig. 1.6 Comparativa pantalles d'inici de dispositius *Android* i *iOS*

Per contra com ja s'ha comentat, les interfícies d'usuari d'*Apple* són homogènies; si saps utilitzar un *iPad* saps utilitzar un *iPhone* i a l'inrevés.

Tornant a *l'iOS*, un altre dels grans encerts ha estat la *AppStore*, el fet de ser un sistema tancat assegura que oficialment només hi ha un lloc on aconseguir les aplicacions per al sistema.

Això assegura tres coses: La primera un gran control de les aplicacions autoritzades sobre la teva plataforma, la segona és que el desenvolupador no s'ha de preocupar de distribuir la aplicació i la darrera és que l'usuari final sap que totes les aplicacions que estan disponibles per al seu terminal estan concentrades al mateix lloc.

La primera característica té dues lectures. La primera, positiva, és que aquest fet minimitza molt la possibilitat d'introduir software maliciós a la plataforma i en cas d'introduir-se, facilita molt la eradicació un cop detectat.

La segona lectura, negativa, és que aquest gran control, monopoli, és regentat per la pròpia *Apple* i la indústria del sector ha reportat moltes queixes de terceres companyies, com per exemple *Adobe* amb el seu software *Flash*, que veuen com les seves aplicacions són rebutjades i per tant no distribuïdes sense motius massa justificats.

Un altre cop, fent el contra anàlisi amb *Android*, el funcionament no és igualt, ans al contrari. A *Android* es permeten pujar qualsevol tipus d'aplicacions sense gairebé cap tipus de filtre, no hi ha una única tenda centralitzada que aglutini totes les aplicacions disponibles –*Android Market*, *Amazon AppStore, Applanet* són només algunes mostres - i a més es poden obtenir i instal·lar aplicacions d'una pàgina web, d'un llapis de memòria, etc.

Per últim, una mancança important que no té la plataforma *iOS* pel fet de ser tancada és el recolzament de la comunitat *OpenSource*. Avui dia en un món cada cop més globalitzat, i més en l'ambit de les TIC, el treball col·laboratiu està demostrant-se molt eficaç i confiable per al desenvolupament i evolució d'una plataforma.

La comunitat que hi ha darrera *d'Android* és molt gran i sense dubte encara que només sigui per el volum de gent implicada darrere de projectes d'aquesta magnitud, sempre avançarà més qui més gent aglutini. *Apple* disposa de la seva extensa plantilla d'enginyers i desenvolupadors però *Google* fa extensible el seu codi a tota la comunitat web.

Tota aquesta informació queda exemplificada a continuació, de forma visual a la taula 1.8.

Taula 1.8 Avantatges i Inconvenients *iOS*

Avantatges	Inconvenients
Software altament optimitzat	Limitació terminals
Compatibilitat absoluta hardware	Sense games econòmiques
Control total sobre la plataforma	Limitació a llibreries de desenvolupament d'*Apple*
No Fragmentació	Obligació d'utilitzar *Objective*-C
Tenda centralitzada d'aplicacions	Sense comunitat *Open Source*
Model negoci rentable per al desenvolupador	Llicència anual per a desenvolupador 4 cops més cara que la *d'Android*.

CAPÍTOL 2. VIDGEOLOCATOR. CREACIÓ D'UNA APLICACIÓ

2.1. INTRODUCCIÓ

Aquest capítol versarà sobre el procés de disseny d'una aplicació per IOS. Entre d'altres, les tasques a realitzar consistiran en identificar les funcionalitats de la aplicació, identificar les tecnologies que haurà d'emprar, dissenyar la arquitectura del sistema i per últim consensuar el comportament que tindrà la mateixa davant dels esdeveniments no previstos, com ara la recepció d'una trucada o un missatge en ple funcionament.

En primer lloc, cal explicar què és i què fa la aplicació: VidGeoLocator.

VidGeoLocator consisteix en un sistema de compartició, localització, reproducció i enregistrament de vídeos.

El funcionament de la aplicació engloba la capacitat de crear o seleccionar un vídeo prèviament enregistrat, afegir-hi dos camps de descripció –un títol i una explicació del contingut del propi vídeo- i pujar-lo al servidor. Posteriorment, mitjançant els serveis de localització i de mapes disponibles al dispositiu, ubicar tots els vídeos disponibles a la base de dades -independentment de l'usuari que els ha pujat al servidor- en un mapa polític i, prèvia selecció d'un vídeo, reproduir-lo. A més a més, com a apartat addicional, es preveu la adopció d'un sistema de Realitat Augmentada -com a alternativa al mapa polític- per a la localització i posterior reproducció dels vídeos.

Tanmateix, només resta afegir que la aplicació estarà dissenyada per un dispositiu en concret, l'iPhone 4.

2.1.1. Especificacions de la aplicació

Per dur a terme una aplicació amb les funcionalitats descrites a la introducció, caldrà la utilització dels frameworks i tecnologies de la Taula 2.1 tan per part del client com per part del servidor:

Taula 2.1 Llistat de tecnologies necessàries a VidGeoLocator

Tecnologies al Client	Tecnologíes al Servidor
Framework de Localització	Creació de *Web Services*
Framework de Càmera	Mapeig d'objectes a una base de dades
Framework de Realitat Augmentada	Pujar dades -Vídeos- al servidor
Framework de *Google Maps*	Base de Dades
Framework de Reproducció multimèdia	
Framework de Comunicació Web	

2.2. DISSENY DEL SISTEMA

VidGeoLocator és un servei que consta d'un servidor on s'emmagatzema la informació i d'una aplicació (client) per visualitzar i actualitzar aquesta informació per *iOS*.

Les aplicacions per *iOS* i en concret per *iPhone* basen tot els seu comportament en esdeveniments que es tradueixen en darrera instancia en accions.

Aquests esdeveniments es generen o bé a traves d'interacció directa usuari-pantalla, gràcies a les capacitats multi tàctils, o bé per esdeveniments generats per la interacció de l'usuari amb els diversos sensors del dispositiu –com l'acceleròmetre alterna entre mode vertical o apaïsat en funció de com es sostingui el terminal-.

A VidGeoLocator es treballarà amb tots dos tipus d'esdeveniments essent el més corrent el primer tipus, el d'interacció usuari-pantalla; clicar directament sobre la acció a realitzar.

Resulta obvi que aquest tipus d'interacció obliga a que totes les accions que es puguin dur a terme mitjançant aquesta aplicació s'hauran de mostrar per pantalla, i aquest fet pot esdevenir un caos si no es gestiona de forma lògica, ordenada i modular.

Per a tal efecte, s'ha dut a terme una estructura dividida en blocs i mòduls de funcionalitat: cada bloc, gestiona una tasca diferent -com la de reproducció de vídeos o la d'enregistrament dels mateixos- i està format per diversos mòduls independents que proporcionen una funcionalitat diferent a aquests mòduls, per exemple: El bloc d'enregistrament de vídeos fa ús d'un mòdul de localització i d'un altre d'enregistrament de vídeo mitjançant la càmera, entre d'altres.

Treballant d'aquesta forma, s'aconsegueix separar tota la lògica del sistema que no té res en comú -com per exemple el *login* de l'aplicació, de la visualització d'un vídeo- i a més a més, s'obtenen mòduls reutilitzables per qualsevol altra aplicació que en un futur pugui fer-ne ús.

Aquesta lògica modular, s'ha aplicat també al que s'anomenen vistes -cadascuna de les diferents possibilitats que es mostren a la pantalla durant l'execució de la aplicació-. D'aquesta forma trobem vistes de registre, de reproducció, de realitat augmentada, etc.

A *l'iOS* es treballa amb dos tipus de vistes, les vistes normals i les vistes modals. Les primeres són les vistes que gestionen el funcionament de la aplicació i del hardware de la aplicació com per exemple el GPS o la API de mapes de *Google*, mentre que les segones són vistes que estan dissenyades per proveir a l'usuari d'una interfície d'entrada de dades o per mostrar dades amb un objectiu merament informatiu, com un panell o una alerta.

La diferència radica més en el funcionament que no pas en la funcionalitat per la que estan dissenyades. Una vista modal es desenvolupa tenint en compte que serà cridada, molt probablement, sobre una altra vista que la requereix –cridi- i, a més a més, tendirà a ser eliminada ràpidament. Per tant, la vista des de la que s'invoca no s'eliminarà de la pila de vistes (concepte que es revisarà al capítol d'implementació del client). De les vistes normals per contra, se n'espera un comportament diametralment oposat.

Feta aquesta introducció, VidGeoLocator permet que un usuari pugui:

- Registrar-se a la base de dades.
- *Loguejar*-se contra la base de dades.
- Veure els vídeos de tots els usuaris al mapa.
- Veure els vídeos de tots els usuaris mitjançant realitat augmentada.
- Enregistrar un vídeo nou.
- Seleccionar un vídeo prèviament enregistrat.
- Penjar un fitxer de vídeo al servidor amb informació sobre posicionament i amb dades que el descriguin al mateix vídeo.

A la figura 2.1, sota aquestes línies es mostra un diagrama esquemàtic, a mode de síntesi, de totes les vistes que intervindran a VidGeoLocator i dels esdeveniments que propicien els canvis entre elles.

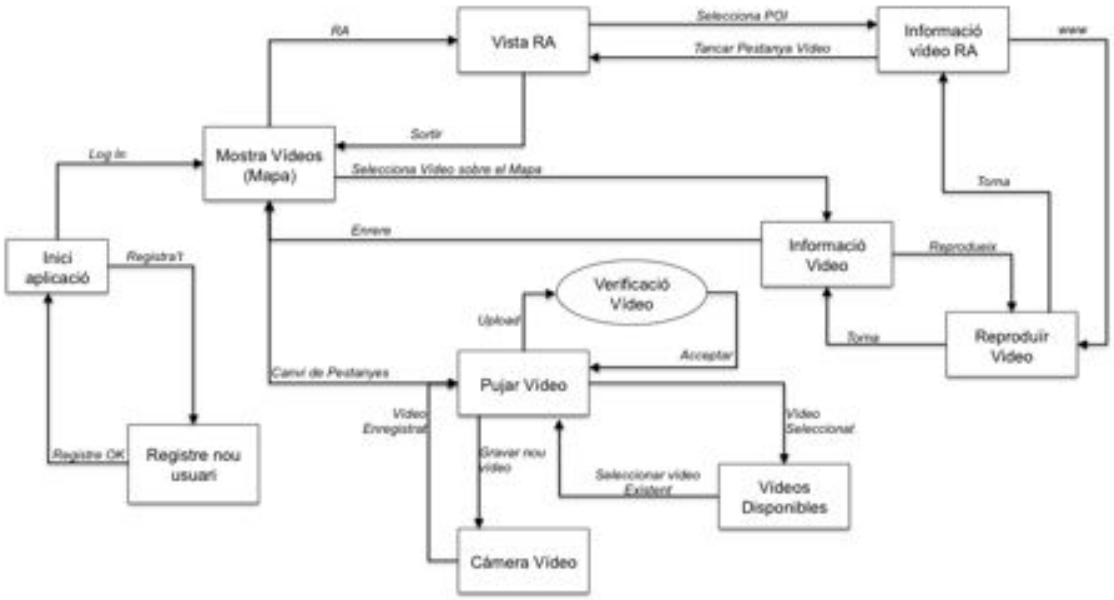

Fig. 2.1 Mapa de vistes de la aplicació

Un cop definida la aplicació, la funcionalitat, les eines necessàries per al seu desenvolupament i dissenyat el mapa de vistes global de la aplicació, el següent pas és dissenyar l'arquitectura del sistema.

2.2.1. Arquitectura del sistema

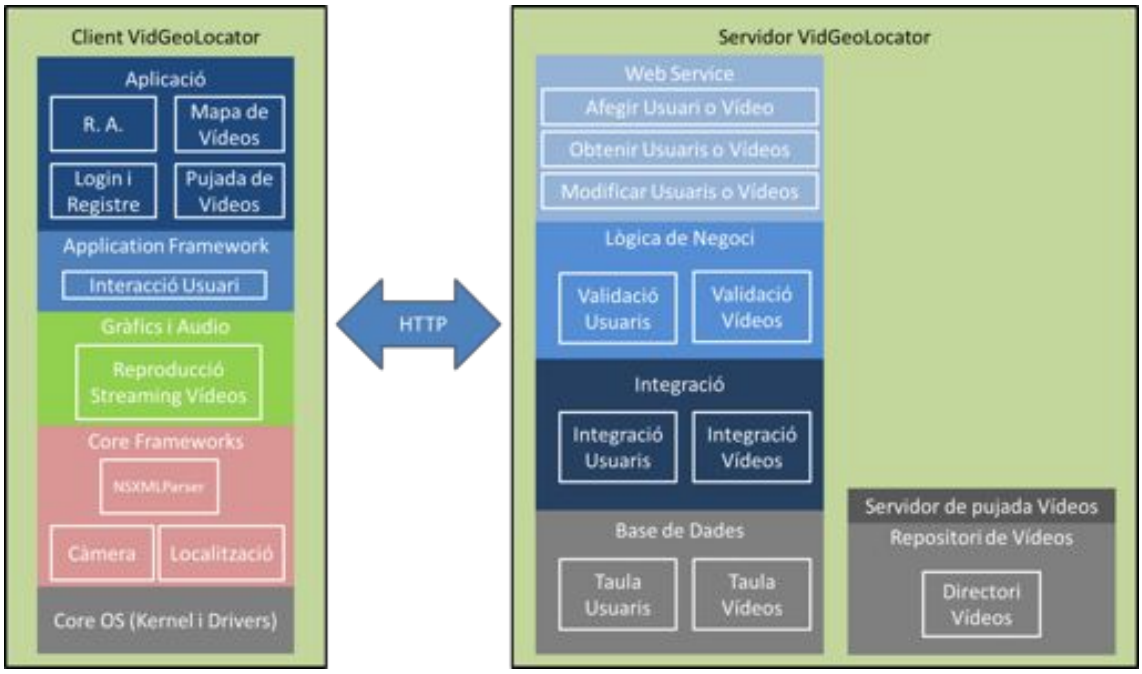

Fig. 2.2 Esquema de l' arquitectura per capes VidGeoLocator

VidGeoLocator es basarà en una arquitectura centralitzada seguint l'esquema client-servidor.

L'esquema de la figura 2.2 està realitzat capa a capa per encabir les tecnologies emprades al client a cadascuna de les capes que conformen l'iOS i que s'han introduït al primer capítol del document i que seran desenvolupades en capítols posteriors.

Pel que fa al servidor, consta de dos serveis independents, un servei principal, que duu el pes de tota la comunicació amb la aplicació mitjançant una interfície *Web Service* –gestiona totes les sol·licituds que rep de l'usuari com ara consultes de registre o peticions de vídeos-, i un segon servei molt més simple que consta d'un *servlet* que emmagatzema en un repositori els vídeos que es pugen des del client.

La decisió de comunicar client i servidor mitjançant un *Web Service* respon al fet que, a canvi d'elaborar una mica més el servidor, s'aconsegueix que client i servidor puguin comunicar-se independentment del llenguatge en que han estat escrits –en aquest cas Java per a la implementació del servidor i *Objective*-C per la del client- utilitzant com a intermediari el llenguatge d'etiquetes XML.

Aquest fet, obre les portes a l'hipotètic desenvolupament de nous clients per qualsevol altre plataforma –independement del llenguatge en què s'hagin d'escriure les aplicacions per a aquestes plataformes-, com per exemple *Android*, sense haver de canviar ni una sola línia de codi de la implementació del servidor.

2.3. INTERACTUANT AMB L'APLICACIÓ

2.3.1. El patró Model-Vista-Controlador

Des del primer moment en que s'inicia el desenvolupament d'una aplicació salta a la vista que el disseny d'una aplicació per a *iPhone* es basa en el patró de desenvolupament Model-Vista-Controlador.

Des de els noms de les plantilles fins al nom dels arxius generats automàticament per l' *XCode* –L'entorn de desenvolupament de *l'iOS* i *OS X*- al crear un nou projecte així ho indiquen, així que per tal de proporcionar un punt d'inici a la programació de la aplicació, s'introdueix a continuació en que consisteix aquest patró de desenvolupament.

El patró de desenvolupament Model-Vista-Controlador és un patró de desenvolupament de software que, bàsicament, separa les dades de la aplicació, de la interfície d'usuari i la lògica de control en 3 components diferents com es veu a la figura 2.3.

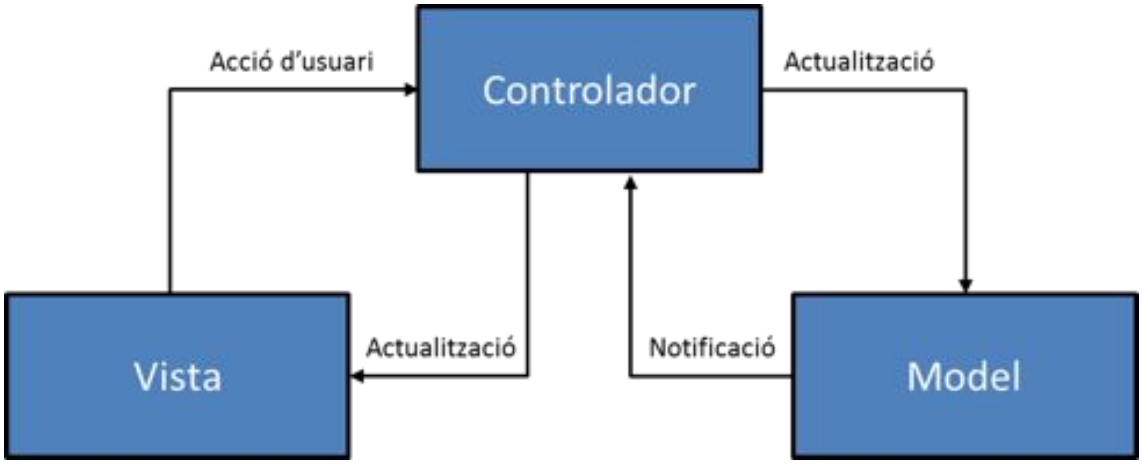

Fig. 2.3 Patró Model-Vista-Controlador

Aquest tipus d'arquitectura és molt utilitzada sobretot en l'àmbit del desenvolupament web, on per exemple, *frameworks* molt estesos com ara *Struts*, de l' *Apache* Software *Foundation*, en fan ús.

La vista és el que l'usuari observa a la pantalla del terminal, el fons de pantalla, les etiquetes, els botons, etc.

El controlador s'encarrega de proveir la vista amb la informació del model i, alhora, actualitzar el model amb la informació que obté de la vista.

Pel que fa al model conté els objectes bàsics de la aplicació que en aquest cas són els usuaris i els vídeos.

Per entendre definitivament com funciona aquest patró, es pren com a exemple un hipotètic cas de registre d'un nou usuari:

Quan un nou usuari es vol registrar a la aplicació, inserta a la vista corresponent les seves dades, bàsicament, correu electrònic, nom d'usuari i contrasenya. Un cop l'usuari ha afegit aquestes dades, clica el botó de confirmació i aquesta acció és capturada pel controlador que en primera instància comprova que es compleix els requisits a nivell local (que no hi hagi cap camp buit) i, posteriorment, envia una sol·licitud de creació d'usuari cap al servidor on estan emmagatzemats tots els usuaris consumint el *Web Service*.

Un cop el servidor hagi acabat de dur a terme el registre, enviarà una notificació cap al controlador a mode de feedback per informar sobre l'estat de la petició i en funció del valor obtingut, el controlador actualitzarà la vista amb un avís d'error o amb una notificació de registre completat.

2.3.2. IBaction, IBoutlet, Protocol i Delegate

En aquesta secció s'introduiran conceptes clau relacionats amb la implementació del codi, necessaris per a la comprensió del mateix en els capítols posteriors i, a més a més, s'exposarà com s'implementa el patró MVC a les aplicacions d'iOS.

Es pot afirmar que al desenvolupament per *iPhone*, hi ha dos grans mòduls independents l'un de l'altre per al desenvolupament d'una aplicació. D'una banda es troba el codi font de la pròpia aplicació i per l'altra el disseny de la interfície.

Pel que fa al desenvolupament de codi -escriptura, depuració i compilació- s'empra l'XCode mentre que pel que fa a la gestió de la interfície gràfica s'utilitza l'Interface Builder.

Així doncs es troben dos blocs diferents gestionats per dues aplicacions diferents. Cóm s'estableix la relació entre el codi i la interfície gràfica? La resposta és mitjançant *IBOutlets* i *IBActions*.

IBOutlet i *IBAction* són dos etiquetes que s'apliquen a determinats objectes alhora de declarar-los al codi de la aplicació. Serveixen per identificar aquells objectes o mètodes que d'alguna forma estan relacionats amb la interfície gràfica de la aplicació, que estan presents a les vistes, tals com botons, quadres de text, imatges, etc; i les accions de l'usuari mitjançant la interfície amb mètodes del codi respectivament.

L'IBOutlet enllaça unidireccionalment un objecte del codi cap a la interfície gràfica, mentre que, pel contrari, l'IBAction enllaça en sentit contrari, de la interfície gràfica cap al codi.

D'aquesta manera, si s'observa una altra vegada la figura 2.3 Patró Model-Vista-Controlador a la pàgina anterior, es relaciona *l'IBAction* i *l'IBOutlet* amb els esdeveniments d'acció d'usuari i refresc respectivament que relacionen la vista amb el controlador.

Un exemple d'us senzill és el següent: es considera una interfície amb una etiqueta de text buida i un botó. Per l'altre costat, pel que fa al codi, es declara un objecte del tipus text amb l'etiqueta *IBOutlet* i s'escriu un mètode omplir text amb l'etiqueta *IBAction*. que té com a funció principal escriure una paraula a *l'IBOutlet* declarat.

Un cop relacionats els dos objectes del codi amb les seves representacions a la interfície gràfica s'executa l'aplicació.

Al polsar el botó a la interfície gràfica, s'executa el codi etiquetat amb *IBAction* que està relacionat amb el botó –acció d'usuari- donant com a resultat que l'etiqueta de text relacionada per *l'IBOutlet* s'omple de contingut -Actualització-.

Un cop explicat el funcionament del patró de vista controlador aplicat al desenvolupament per *iOS*, només resta explicar dos conceptes *d'Objective*-C molt utilitzats a l'aplicació: els protocols[3] i els delegats[4].

Es diu que una classe adopta un protocol quan aquesta implementa tots els mètodes obligatoris –si porten l'etiqueta @*required*- que proveeix aquest protocol. Té una equivalència directa a Java, les implementacions.

Un cop s'ha assignat, implementat e inicialitzat un protocol, aquest realitza una sèrie de tasques en segon pla de forma transparent al desenvolupador.

La veritable utilitat del protocol resideix en aquestes tasques transparents al desenvolupador: Un cop el protocol ha finalitzat una d'aquestes tasques que realitza de forma transparent enviarà una notificació a la classe que l'hagi demanada, la que implementi el delegat.

Un delegat és un objecte que actua en el lloc d'un altre, d'aquesta manera, quan una aplicació implementa un protocol i assigna un delegat, el que està passant és que protocol realitzarà una sèrie de tasques de forma autònoma i un cop les hagi finalitzades, si un objecte li sol·licita – i aquest objecte implementa els mètodes apropiats del protocol i es declara com el delegat d'aquest protocol- enviarà una notificació que llençarà un dels codis implementats del propi protocol.

De forma simplificada, si es considera el *framework* de captura d'imatges, el funcionament és el següent: Es tria una classe que adopti el protocol de captura d'imatges. Un cop s'adopta el protocol s'inicialitza i s'indica quina classe –una instancia d'aquesta classe - serà la delegada del protocol. Un cop s'ha assignat el delegat del protocol, només resta implementar els mètodes del protocol desitjats (i per suposat dels obligatoris) que el desenvolupador vulgui realitzar, suposem en aquest cas realitzar una fotografia.

Un cop l'aplicació està en funcionament la successió de fets és la següent: Primer s'inicia l'aplicació i es carrega la vista de la càmera, l'usuari pren una fotografia i el protocol s'encarrega de la captura de forma transparent al desenvolupador. Un cop s'ha realitzat la captura i ja s'ha obtingut la imatge, el

protocol envia una notificació a la classe que l'adopta i aquesta notifica al delegat corresponent –en aquest cas ella mateixa, tot i que no és obligatori- mitjançant l'execució del mètode corresponent, en aquest cas seria el mètode anomenat capturaRealitzada.

Així doncs, a percepció del desenvolupador, tots els mètodes implementats d'un protocol executats mitjançant delegació seran cridats de forma asíncrona.

2.4. CICLE DE VIDA D'UNA APLICACIÓ

Abans de començar el desenvolupament d'una aplicació cal tenir clar quin és el funcionament d'aquesta dins la plataforma que l'executa. Cal saber com actua la plataforma davant certs esdeveniments que es puguin donar com per exemple una trucada mentre una aplicació s'està executant, així com saber si com a desenvolupador es pot triar com actuarà la aplicació davant d'aquests esdeveniments.

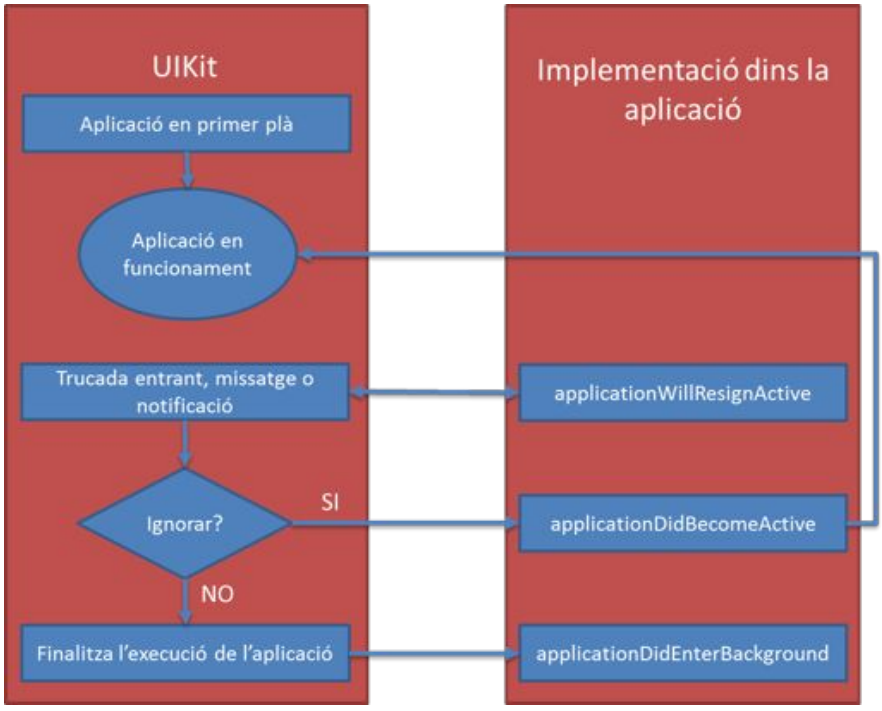

Fig. 2.4 Cicle de vida d'una aplicació a *l'iOS*

A la figura 2.4 es veu l'esquema bàsic de funcionament d'una aplicació. Inicialment, *l'UIKIT –User Interface Kit-* gestiona l'entrada de pulsacions tàctils sobre la pantalla, proporcionant la coordenada exacta on s'ha polsat la mateixa.

Si detecta que l'usuari clica –o polsa- sobre una aplicació, llençarà la mateixa a primer pla. És important notar que no l'inicia ja que des de la implantació de la multitasca, una aplicació que no s'ha executat mai no es distingeix d'una aplicació que esta en segon pla suspesa ja que cap de les dues està executant cap línia de codi.

Un cop la aplicació està a primer pla es comprova si s'ha d'iniciar (aplicació no s'estava executant) o si ja existeix una instancia de la mateixa (aplicació s'ha suspès prèviament) i es recupera. En qualsevol dels casos la aplicació passarà a estar activa i des d'aquest moment l'usuari tindrà control total sobre la mateixa. Mitjançant el bucle d'esdeveniments es pot navegar per la mateixa i utilitzar-la.

Si mentre l'usuari utilitza la aplicació es produeix un esdeveniment extern a la pròpia aplicació, com una trucada, un missatge o una notificació de calendari, immediatament la aplicació s'aturarà i cridarà al mètode anomenat *applicationWillResignActive*.

Aquest mètode dona al desenvolupador la oportunitat de realitzar totes les tasques de protecció d'informació necessàries com podria ser desar les activitats de l'usuari, emmagatzemar les preferències de la aplicació o la partida de l'usuari si l'aplicació es tractés d'un joc.

A més a més durant aquest procés, la aplicació queda en un segon pla per remarcar i fer notar a l'usuari la interrupció que s'està produint i deixant-li triar si vol interactuar amb la interrupció o si vol ignorar-la.

En cas que vulgui interactuar amb la mateixa, poden passar dues coses, la primera és que la aplicació suporti la multitasca i la segona és que no la suporti.

Si la aplicació està preparada per a la multitasca, aquesta serà enviada a segon pla i entrarà en estat de repòs tal i com s'ha explicat. La última acció de la aplicació és executar el mètode *ApplicationDidEnterBackground* en què el desenvolupador pot realitzar tasques de neteja tals com tancar connexions

L'altre opció és que la aplicació no suporti la multitasca, com als models antics de l'iPhone. En aquest cas la aplicació es tancarà mitjançant la crida al mètode *ApplicationWillClose* que de la mateixa forma com actua el mètode *ApplicationDidEnterBackground*, dóna al programador la oportunitat de tancar totes les connexions que pugui tenir actives la aplicació abans de tancar-la.

D'altra banda, si l'usuari decidís no fer cas de la alerta, trucada o esdeveniment que hagués interromput l'execució de la aplicació, l'aplicació passarà de l'estat de repòs ja comentat a l'estat actiu -mitjançant la crida al mètode *ApplicationWillBecomeActive* (que també s'executa quan una aplicació passa de l'estat de repòs de la multitasca a restar activa)- reactivant connexions o qualsevol altre tipus d'inicialització que necessiti la aplicació i que havien estat aturades en el moment de la interrupció.

Per acabar, un cop exposats els dos possibles comportaments d'una aplicació a iOS, s'ha decidit que VidGeoLocator s'implementi com una aplicació amb suport per la multi tasca. Per tant, a no ser que l'usuari forci la sortida de la aplicació, aquesta sempre restarà en segon pla –en estat de repòs- quan no s'utilitzi. Arribats a aquest punt, ja queda completament definida la aplicació, només restarà implementar tant client com servidor.

CAPÍTOL 3. DESENVOLUPAMENT DEL SERVIDOR

En aquest capítol, es tractarà la implementació proposada per a la realització de la aplicació VidGeoLocator basada en el disseny, requeriments i arquitectura exposats a l'apartat anterior. D'aquesta manera es recolliran tots els aspectes relacionats amb els dos serveis que composen el costat del servidor.

L'estructura del capítol és molt simple: d'una banda es troba la implementació del servei principal de la aplicació –amb explicació de cadascun dels elements que el composen- i d'altra d'un mode molt més conceptual es troba el segon servei de la aplicació que s'encarrega de la pujada de vídeos al servidor.

3.1. SERVEI PRINCIPAL DE LA APLICACIÓ

A la figura 3.1, es representa visualment quines tecnologies s'han emprat per implementar cadascun dels mòduls del servidor. S'aprecia com la interfície de Web Service ha estat implementada mitjançant *Apache Axis2*, la lògica de negoci (i de tota la aplicació) està escrita mitjançant *Java*, la capa d'integració es basa en el *framework Hibernate* i per últim la base de dades és *MySQL*.

En aquest apartat, es tractaran cadascun d'aquests mòduls per separat per compendre el funcionament del servei.

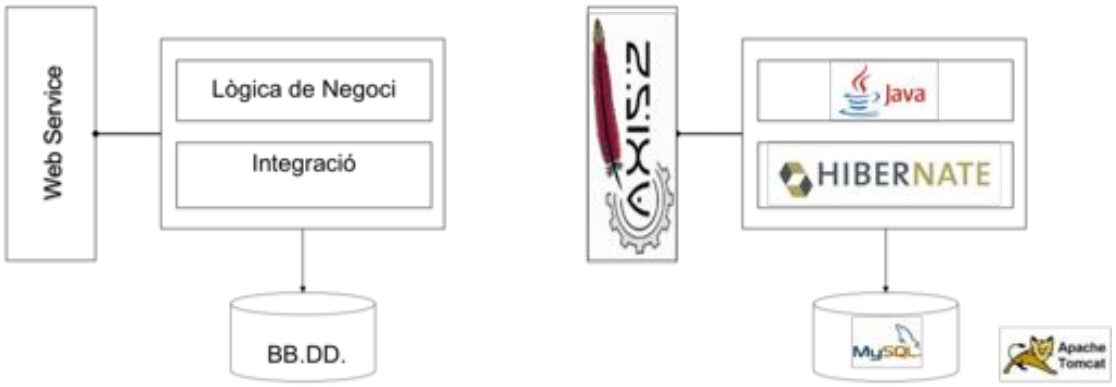

Fig. 3.1 Tecnologies emprades per a la implementació del primer servei

La implementació del servidor està muntada sobre un servidor *Apache Jakarta Tomcat* 6 en una màquina –virtualitzada- amb un sistema operatiu *UNIX*, concretament *Ubuntu* 9.10.

3.1.1. Web Service

Ja s'ha comentat amb anterioritat els avantatges obtinguts comunicant client i servidor mitjançant una interfície *Web Service* però no s'ha definit que és un *Web Service.*

Un servei web -*Web Service*- consta d'un conjunt de protocols i estàndards que serveixen per intercanviar dades entre aplicacions. Diferents aplicacions de software desenvolupades en diferents llenguatges de programació, i

executades sobre qualsevol altre plataforma, poden utilitzar els serveis web per a intercanviar dades entre elles mitjançant xarxes d'ordinadors, com ara internet.

El motiu de la elecció d'*Axis2* com a *Web Service* és simple, és l'únic que està dissenyat per funcionar sobre *Apache Jakarta Tomcat* -el servidor sobre el que corre el sistema- i, a més a més, pot proporcionar *Web Services* de tipus *RESTful [5]*.

Fig. 3.2 Resposta del Web Service generada per petició *HTTP get*

A la Fig. 3.2, es representa la invocació d'un *Web Service* mitjançant la seva URL, (que generarà una petició *HTTP get*) directament executada des d'un navegador i la seva corresponent resposta en format XML.

Com s'aprecia a la resposta torna la definició d'un objecte sencer, en aquest cas un vídeo.

D'aquesta manera quan el client (o el terminal que consumeix el *Web Service)* rep la resposta, només precisa de dues coses per poder obtindré la informació, el model d'objecte que esta rebent, i un *parsejador* d'XML per poder-ne extreure els valors.

3.1.2. Lògica de Negoci

La capa de lògica de negoci s'encarrega de coordinar totes les operacions que es donen en aquest servei a la part del servidor. És el mòdul que s'encarrega de processar les peticions que arriben al servidor mitjançant la interfície del *Web Service*, validar-les i en ultima instancia transmetre-les a la seva capa immediatament inferior -la capa d'integració- perquè aquesta pugui comunicar-se amb la base de dades i així transmetre les dades de la petició rebuda.

Tanmateix cal observar que la comunicació que estableix entre ambdues capes és bidireccional ja que també s'encarregarà de retornar en sentit contrari (de la capa d'integració al *Web Service*), la resposta a la petició efectuada.

Aquesta capa realitza a més una funció addicional, proporciona un mínim sistema de control de flux.

Mentre les altres capes serveixen per executar operacions bàsiques sobre els diferents objectes amb els que interactuen -afegir, modificar, carregar o eliminar- ja sigui un usuari o un vídeo, la capa de lògica de negoci s'encarregarà d'enviar les peticions rebudes mitjançant la interfície del *Web Service* cap a la capa d'integració i retornar les respostes que aquesta última obtingui en sentit contrari, fent com es comenta, fent de nexe entre ambdues capes.

D'aquesta forma coordina i comunica les dues capes sense necessitat que cap d'elles tingui coneixement de cóm està implementada l'altre, afavorint la modularitat del sistema al llarg de tot el servei.

La segona de les característiques que proporciona la capa de lògica de negoci és la de control de flux. Consisteix en assegurar-se que cada petició que rep mitjançant la interfície Web Service es pot realitzar satisfactòriament abans de passar-la cap a la capa d'integració, mitjançant unes peticions de consulta prèvies per evitar realitzar operacions de lectura i/o escriptura invàlides sobre la base de dades.

A la Figura 3.3 es troba un exemple que mostra la hipotètica petició d'afegir un usuari al servei.

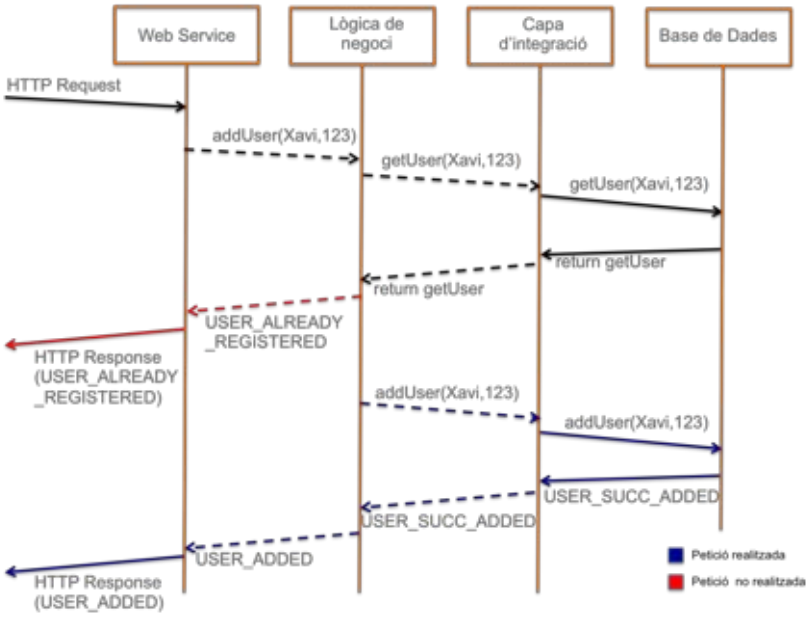

Fig. 3.3 Diagrama de flux petició-resposta

Un exemple de funció no vàlida podria ser la creació d'un nou usuari amb unes dades que ja existeixen al servidor:

Quan l'usuari executa la crida de la funció *crea usuari*, el *Web Service* entrega la petició a la capa de lògica de negoci i aquesta s'encarrega de comprovar que a la base de dades no existeixi un usuari amb aquestes dades abans de validar la operació.

És aquí on s'obren dues possibilitats: Si no existeix cap usuari amb les mateixes dades (Fig. 3.3 Diagrama de flux petició-resposta, petició realitzada), enviarà l'ordre d'afegir un usuari a la capa d'integració amb la certesa que no hi haurà cap problema d'incompatibilitat i, un cop afegit l'usuari, informarà a la interfície de Web Service. Per contra, si al realitzar la comprovació topa amb un usuari amb el mateix nom (Fig. 3.3, Diagrama de flux petició-resposta, petició no realitzada) informarà directament a la interfície Web Service del motiu pel qual aquesta petició no ha pogut ser atesa (en aquest cas, no pot haver-hi dos usuaris amb el mateix nom).

Finalment, el Web Service codificarà en format *XML* la resposta que comunicarà al client- via HTTP *Response*.

3.1.3. Integració

Per a la capa d'integració, com ja s'ha *adelantat*, s'ha fet ús del *framework Hibernate*.

Hibernate és una eina de Mapeig *objecto-relacional* desenvolupada per a la plataforma Java que facilita el mapeig d'atributs entre una base de dades relacional tradicional i el model d'objectes d'una aplicació mitjançant arxius de configuració *XML* o bé –com en aquest cas- anotacions en les entitats que permeten establir les relacions.

El fet d'emprar el *framework* d' *Hibernate*, té el gran avantatge d'abstreure al programador de barrejar el codi font de la seva aplicació amb les crides SQL.

Quan es diu que és una eina de mapeig objecto-relacional significa exactament que és capaç de convertir directament les classes –objectes- seleccionades sobre qualsevol tipus de base de dades –relacions- que suporti el propi framework, sense haver de reescriure codi, simplement canviant l'arxiu de configuració de la base de dades.

Al quadre de la pàgina següent, es mostra el fitxer *xml* de configuració mínim pel funcionament del *framework d'hibernate*. Cal fixar-se especialment en dos tipus de paràmetre: els *property* i els *mapping.* Mentre que el primer s'encarrega de configurar paràmetres relacionats amb l'accés a la base de dades (direcció, nom de la base de dades, usuari i contrasenya), el segon, com el seu nom indica, s'encarrega de dir quines seran les classes que haurà de mapejar contra la base de dades.

```
<hibernate-configuration>
<session-factory>
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://127.0.0.1:3306/test</property>
<property name="connection.username">root</property>
<property name="connection.password"></property>
<property name="connection.pool_size">2</property>

<property name="hibernate.c3p0.min_size">1</property>
<property name="hibernate.c3p0.max_size">10</property>
<property name="hibernate.c3p0.timeout">1800</property>
<property name="hibernate.c3p0.max_statements">50</property>

<mapping class="models.User" />
<mapping class="models.Video" />
</session-factory>
</hibernate-configuration>
```

Aquest fitxer dóna una idea de les classes que s'empren a aquest projecte – *Video* i *User*-. A primera vista podria semblar que qualsevol classe és susceptible de ser *mapejada* però aquesta afirmació no seria certa.

Per que una classe pugui ser *mapejada* amb *hibernate* cal que sigui una classe POJO (*Plain Old* Java *Object*). Una classe *POJO* és una classe que només posseeix atributs i mètodes per accedir als mateixos –el que s'anomena *getters* i *setters*. A més a més, *hibernate* requereix que com a mínim, la classe tingui un constructor sense paràmetres.

A l'Annex B es troba informació detallada del framework d'hibernate així com la implementació de la classe *POJO* d'usuari.

3.1.4. Bases de Dades

La base de dades emprada, és una base de dades *MySQL*. La elecció es basa en criteris purament personals, és la base de dades utilitzada durant la carrera i a més a més és compatible amb *hibernate* així que no s'han comprovat altres possibles alternatives.

Durant la etapa de disseny es va pensar en realitzar un esquema com el de la figura 3.4, es crearien les entitats d'usuari i de vídeo juntament amb una taula relacional entre ambdues per poder obtenir la informació completa de forma bidireccional: Per una banda, seleccionant un usuari es podrien obtenir tots els seus vídeos i per l'altra es podria obtenir l'usuari creador d'un vídeo directament a partir d'un vídeo.

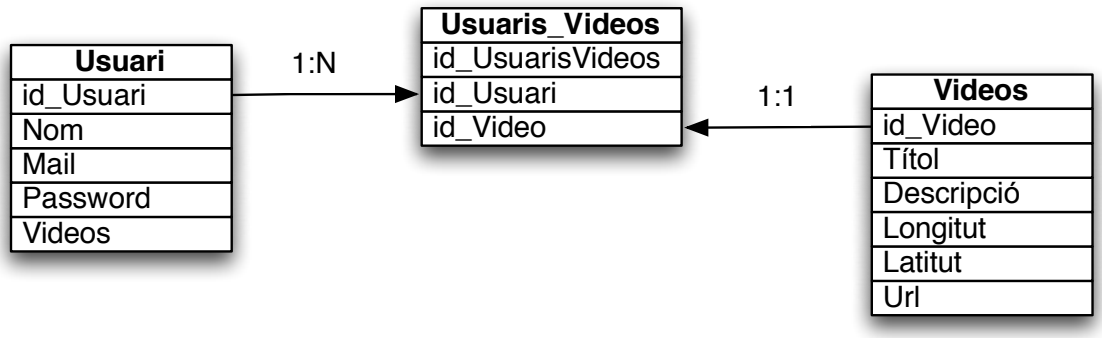

Fig. 3.4 Diagrama Entitat-Relació

La idea era vàlida però de seguida es va veure que no era un plantejament gens òptim degut a la estructura creada. El problema radica en el fet d'utilitzar un *Web Service RESTful*.
Com ja s'ha comentat anteriorment, el *Web Service* retorna un XML amb el contingut complet de l'objecte. Això vol dir que cada cop que es volgués obtenir una llista amb tots els usuaris, com que els vídeos són un atribut més de cada usuari, haurien de ser obtinguts també, obtenint com a resposta, en definitiva, un XML amb tota la base de dades sencera.

VidGeoLocator no seria massa escalable, així que es va decidir buscar una alternativa.

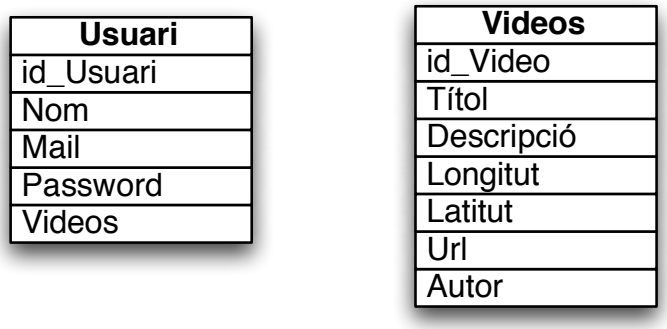

Fig. 3.5 Diagrama d'entitats final

Finalment es planteja com a solució utilitzar dues taules independents l'una de l'altre tal i com es reflexa a la Figura 3.5 una per a la gestió d'usuaris i l'altre per a la gestió de vídeos.

D'aquesta manera, totes les peticions d'accions relatives a la administració d'usuaris (afegir, comprovar, editar o eliminar) serà gestionat completament per la taula d'usuaris mentre que, anàlogament, la taula de Vídeos serà qui s'encarregui de gestionar totes les peticions referents a els vídeos de tots els usuaris.

3.2. SERVEI DE PUJADA DE VÍDEOS

Aquest és el segon servei de la aplicació, el servidor de pujada de vídeos, el seu funcionament és molt bàsic i està basat en un API d'Apache Commons anomenada FileUpload que gestiona gairebé de forma automàtica la pujada de fitxers a un servidor.

A continuació a la figura 3.6 es veu el funcionament del mateix:

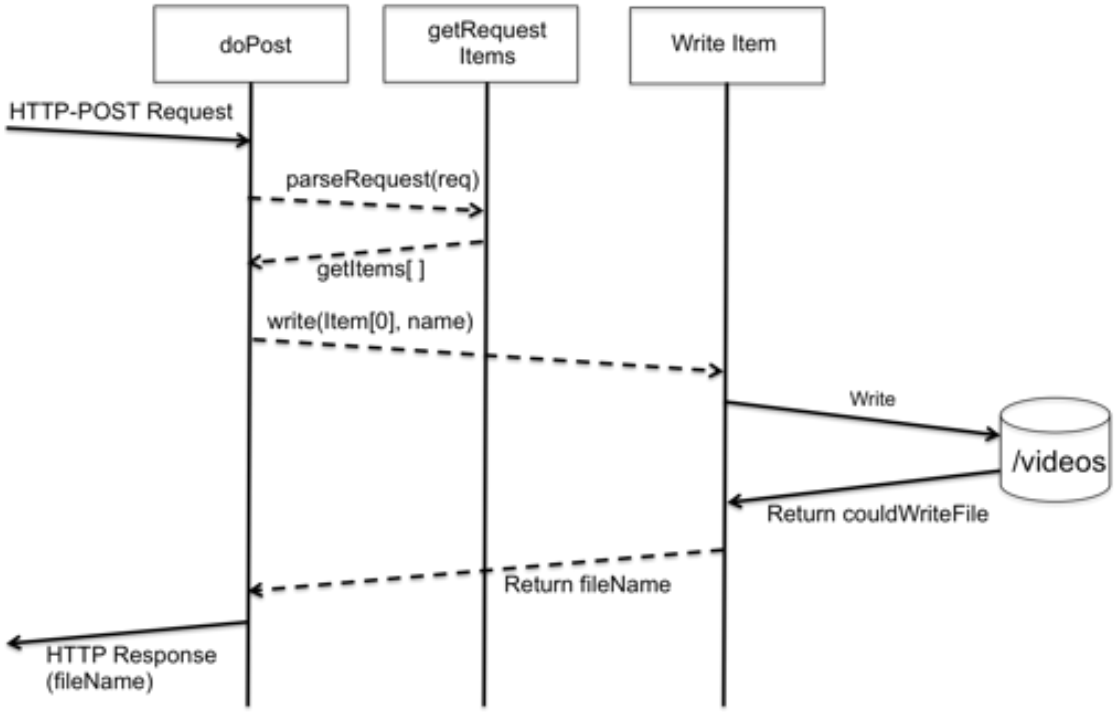

Fig. 3.6 Procés de pujada d'un fitxer al servidor

Quan una petició –*HTTP post*- arriba al *servlet* passa immediatament a ser *parsejada* per tal de poder obtenir les dades i el nom del fitxer de la mateixa.

Un cop obtinguda la informació, es procedeix a escriure les dades a un fitxer amb el nom obtingut situat a la carpeta /vídeos dins del servidor.

En cas que tot les operacions s'hagin produït satisfactòriament es generarà una resposta *HTTP* que contindrà únicament el nom del fitxer escrit.

El fet de tornar aquest paràmetre és útil bàsicament com a *ACK* per poder comprovar que s'ha pujat correctament el vídeo al servidor.

Pel que fa a la detecció de noms duplicats, aquest servidor no ho comprova. Aquestes consideracions es prenen al costat del client i es comentaran al capítol corresponent però bàsicament, el que es fa és enviar com a nom del fitxer el resultat de realitzar una funció de *hash -MD5* [6]- del video en qüestió.

CAPÍTOL 4. DESENVOLUPAMENT DEL CLIENT

En aquest apartat es tractarà el desenvolupament a nivell més tècnic sobre la implementació del client per a *iOS* de VidGeoLocator, tractant aspectes com la gestió de la memòria o els components que conte cadascun dels mòduls que composen la aplicació.

No obstant aquestes consideracions, l'objectiu principal d'aquest capítol és definir i explicar els diferents mòduls que composen la aplicació.

4.1. IMPLEMENTACIÓ DE L'APLICACIÓ

Tal i com s'ha establert al capítol 2, per desenvolupar la aplicació es partirà del concepte de blocs i mòduls. Un bloc és el conjunt format per vistes, controladors i, en general, qualsevol altre element que aquest pugui necessitar –com *frameworks*- que conjuntament estan dissenyats per realitzar una funcionalitat. Els mòduls seran cadascun dels elements independents que composen els blocs i que tenen una funció específica dins dels mateixos.

Si s'observa la figura 4.1 a continuació, es veu una versió simplificada del Mapa de vistes de la aplicació (Fig. 2.1), en el que s'aprecia de forma clara els diferents mòduls que disposen d'una vista individual i que composen la aplicació.

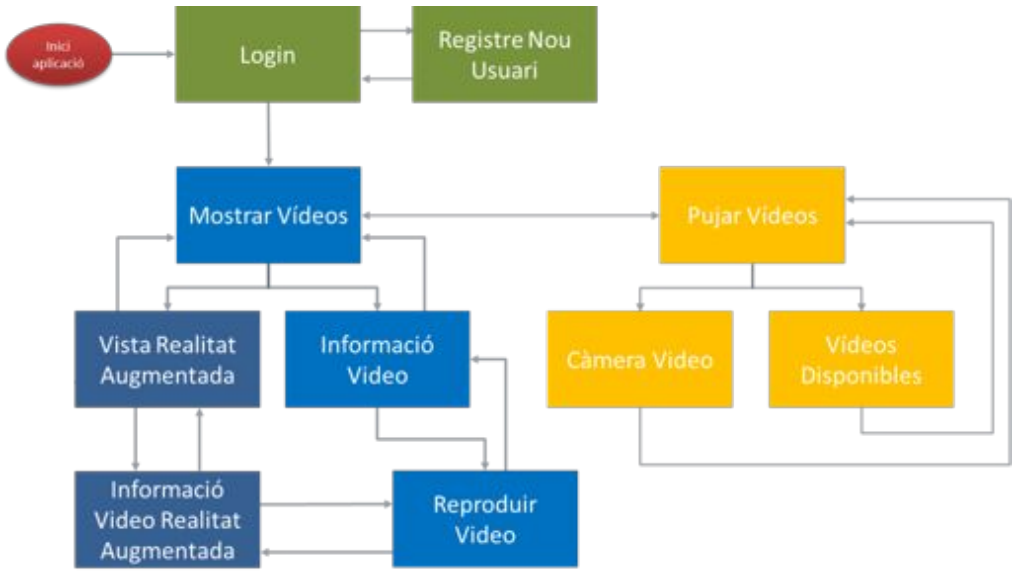

Fig. 4.1 Disseny simplificat de vistes de la aplicació

Els diferents colors que composen la figura es relacionen amb els blocs als que pertanyen, de forma que, visualment, resulta molt intuïtiu esbrinar els mòduls que guarden relació entre ells.

D'aquesta manera es pot distingir entre 4 blocs independents que s'encarreguen de gestionar el funcionament de la aplicació.
Els blocs a tractar seran:

- Registre i Autenticació
- Mostrar Vídeos
- Pujar Vídeos
- Realitat Augmentada

A continuació s'exposaran cadascun d'ells de forma separada indicant entre d'altres la seva funcionalitat, les tecnologies que utilitzen, i si s'escau una explicació de les diferents APIs utilitzades per dur a terme les funcionalitats.

4.1.1. Registre i Autenticació

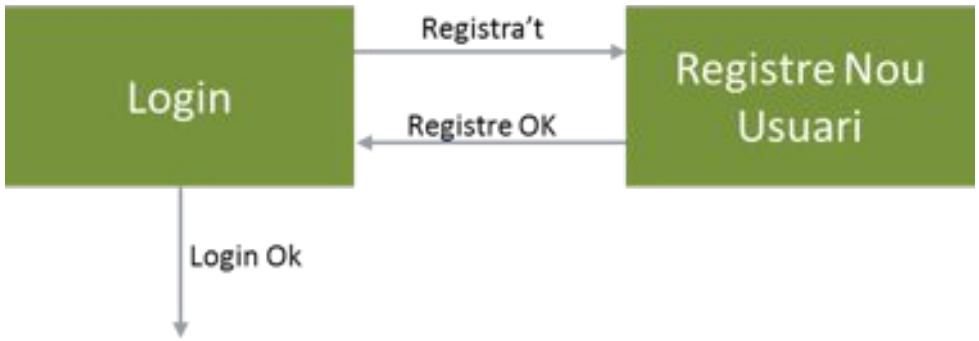

Fig. 4.2 Bloc aïllat de Registre i Autenticació

La funcionalitat d'aquest bloc s'explica per si mateixa simplement amb el nom. La principal funció d'aquesta és proporcionar a un usuari dels elements necessaris per *logejar*-se -autenticar-se- i poder així accedir a la aplicació.

Tanmateix, també contempla que un usuari no registrat tingui la opció de registrar-se a la base de dades de VidGeoLocator per poder fer-ne ús posteriorment.

Les tecnologies emprades en aquest bloc són molt bàsiques ja que no cal executar cap tipus de lògica al client. L'únic element que s'ha de tenir en compte és el fet de connectar-se al servidor de VidGeoLocator per poder contrastar amb la base de dades si el *login* és o no vàlid, i lògicament ser capaç d'interpretar la resposta.

Per gestionar aquesta funcionalitat tan sols cal crear una instancia del *parsejador* d'XML –*NSXMLParser*- amb un constructor específicament designat per *parsejar* el contingut d'una petició *HTTP get*.

```
parser = [ [NSXMLParser alloc] initWithContentsofURL: [ [NSURL alloc] initWithString:url] ];
```

parser és un punter del tipus *NSXMLParser*. El que es fa en aquesta sentencia és crear una instancia de *NSXMLParser* –[*NSXMLParser alloc*]- amb el constructor *initWithContentsofURL*: que espera com a paràmetre d'entrada un objecte NSURL –la URL del *Web Service* amb els paràmetres del nom d'usuari i contrasenya- i assignar-la al punter.

Un cop inicialitzat el punter simplement cal llegir el valor obtingut de la petició i actuar segons hagi estat una petició vàlida o no.

Si el resultat és invàlid, es generarà un *popup* d'alerta –*UIAlertView*- per proporcionar a l'usuari feedback sobre la seva petició -independentment del resultat-. Mentre la petició s'està atenent es mostra per pantalla una vista amb un altre *popup* per indicar que la petició s'està processant.

D'altra banda, si finalment la petició és vàlida les següents crides descarregaran la vista actual del controlador i mostraran la vista principal de la aplicació, que es troba a sota[1].

```
[ [self view] removeFromSuperview];
[self release];
```

És important tornar a remarcar que la memòria en les aplicacions en *Objective-C* la gestiona el desenvolupador i per tant s'ha d'encarregar d'alliberar-la quan convingui. Com en aquest cas, la vista de *login* no tornarà a ser carregada mai més mentre la aplicació estigui en funcionament, es pot alliberar l'espai que ocupa en memòria sense cap problema.

L'altre funció que té aquest primer bloc és la de registrar a un nou usuari. Aquesta funció empra les mateixes eines que el *login*, ja que, procedint de la mateixa forma, cal verificar contra la base de dades que, en aquest cas, no existeixi prèviament un usuari amb les mateixes dades per evitar conflictes[2] entre usuaris.

A la figura 4.3 a la pàgina següent s'aprecia el diagrama de flux del bloc de registre d'un nou usuari. El disseny és molt simple des del punt de vista del client:

Cal tenir present que *l'iOS* és una arquitectura orientada a esdeveniments i en aquest cas qui desencadena el primer esdeveniment és l'usuari al fer clic al botó crear usuari.

Aquest botó té associada una acció que s'encarrega de crear una instancia del *parserXML* tal i com s'ha mostrat abans. Aquesta instància contindrà en la

[1] Al desenvolupament de *l'iOS* només es troba una vista activa –capaç de respondre als inputs de l'usuari- alhora. Tanmateix poden sobreposar-se vistes l'una sobre l'altre en una pila LIFO de forma que, per defecte, la última en afegir-se a la pila és la vista activa. En cas d'eliminar la vista activa, passarà a activar-se la vista immediatament inferior.

[2] La lògica del registre s'ha desenvolupat al costat del servidor per tal d'assegurar que si en un futur altres desenvolupaments fessin ús de la mateixa aplicació aquesta no quedés compromesa per desenvolupadors tercers que no prenen aquestes precaucions

petició *HTTP get* els camps nom d'usuari, contrasenya i correu necessaris per al servei web que està escoltant al servidor.

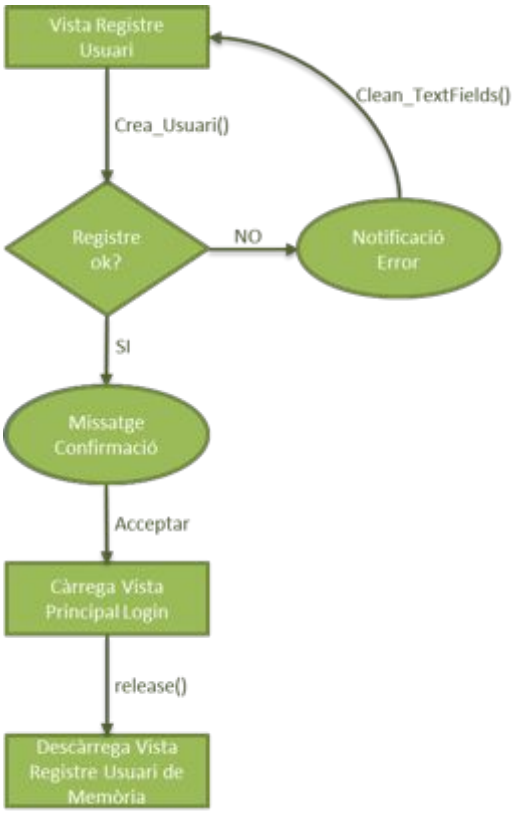

Fig. 4.3 Diagrama de flux del registre d'usuari

Quan la resposta de la petició arribi al client de nou amb el valor de la mateixa es comprovarà si es vàlida o no.

En el cas negatiu, es mostrarà a l'usuari un missatge amb el motiu de l'error, netejarà les dades que l'usuari havia entrat prèviament per registrar-se i a continuació tornarà a la vista de registre per tal que pugui repetir el procés.

Contràriament, si el registre ha estat satisfactori es procedeix a informar a l'usuari de la mateixa manera, mitjançant un *popup* o *UIAlertView*, i un cop l'usuari tanca aquest avís es mostra de nou la pàgina de *login* –es posa a la primera posició de la pila de vistes-.

Un cop la vista de login ha estat completament carregada de nou, es procedeix a descarregar de la memòria la vista de registre, que presumiblement no es tornarà a utilitzar.

En resum:

- El primer bloc de la aplicació consta de dos mòduls independents però gairebé idèntics en funcionament i gestió.

- El funcionament és gairebé el mateix ja que ambdós mòduls només s'encarreguen de fer al servidor una petició *HTTP get* i esperar-ne la resposta per actuar en funció de la mateixa.

- Pel que fa la gestió de les vistes, són vistes modals carregades sobre la vista de mapes, així que tan bon punt han complert amb la seva escomesa són eliminades.

4.1.2. Realitat Augmentada

Aquest mòdul s'encarrega de proporcionar una vista alternativa per visualitzar els vídeos que estan emmagatzemats al servidor.

La realitat augmentada (RA) és un terme que s'aplica a la capacitat de visualitzar l'entorn físic -el món real- amb elements afegits mitjançant la informació sensorial generada virtualment a través d'un ordinador, com sons o imatges.

El concepte de realitat augmentada s'aplica a VidGeoLocator mitjançant tres elements, el GPS, la Càmera anterior del terminal i la Brúixola.

Per dur a terme aquest mòdul, s'ha emprat una API existent desenvolupada per *iOS* i per *Android* anomenada Wikitude [7].

Wikitude proporciona un conjunt d'eines que proveeixen a la aplicació que en fa ús de la capacitat d'afegir una vista amb realitat augmentada. Per fer-ho necessita únicament un requisit: el dispositiu que faci us de la aplicació ha de tenir brúixola (això limita els terminals a l'*iPhone* 3GS i a l'*iPhone*4).

Acomplert aquest requeriment (VidGeoLocator funcionarà sobre *iPhone*4), només cal proveir a la aplicació dels objectes que han de mostrar-se per pantalla: Els *POI*.

Un *POI* és un *Point Of Interest*. Un *POI* és una classe que defineix títol, descripció, coordenades i URL d'un objecte. Si es mira la estructura de la taula vídeos a la base de dades definida al servidor (Fig. 3.5), s'observa que aquesta estructura és perfectament compatible amb la definició d'un *POI.* D'aquesta forma i simplificant, un *POI* equivaldrà a un Vídeo.

Si s'estudia la següent captura s'explicarà gràficament quin és el funcionament d'aquesta tecnologia i cóm es relacionen aquests tres elements.

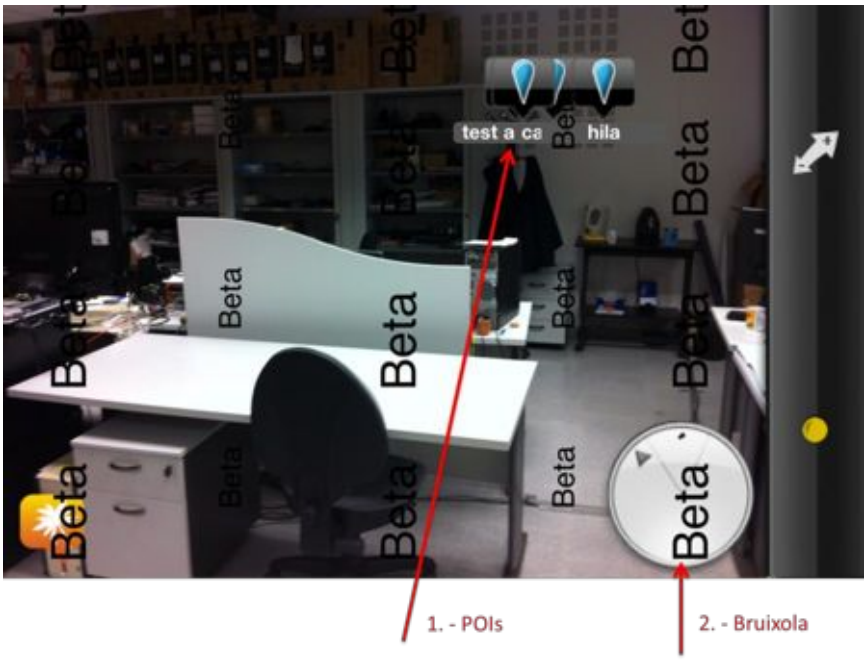

Fig. 4.4 Captura de VidGeoLocator en vista de Realitat Augmentada

L'ús de la càmera proporciona la visió de la realitat a la nostra aplicació. Per fer-ho, la API de *Wikitude* utilitza una vista modificada –sense controls d'enregistrament- de la captura d'imatges de l'*iPhone*.

A sobre d'aquesta capa s'introdueixen dos elements. El primer element és una llista dels *POIs* que són visibles en aquest moment; per saber quins *POIs* són visibles a cada instant cal una sistema que ho pugui controlar. Aquest sistema és la brúixola.

Mitjançant la brúixola, es pot obtenir la direcció de l'usuari respecte a una posició de referència –el nord magnètic de la terra-. Si aquest sensor es combina amb el GPS, s'obté un sistema capaç de localitzar la ubicació i saber la direcció cap a on esta mirant l'usuari[3].

Si a aquests dos conceptes s'afegeix el fet que el camp de visió humà és cònic i, que per tant, només es poden veure els objectes –la realitat- que es troben dins d'aquest con –i a una distancia màxima acotada-, s'obté un sistema per posicionar a la pantalla els objectes –vídeos en aquest cas- en funció de les seves coordenades: Es calcula per cadascun dels *POIs* la seva posició relativa al con de visió, si el POI es troba inscrit dins el con es mostrarà, sino no.

Tot aquest sistema està automatitzat per la API i representat per un sistema de navegació (al punt 2 de Fig. 4.4) . El seu funcionament és el que dóna sentit a la vista.

[3] Realment encara que es parla de localització d'usuari i orientació d'usuari, es més correcte parlar de localització del terminal i orientació del terminal ja que són les seves coordenades i la seva orientació les que s'obtenen.

La brúixola, emulant el seu comportament a la vida real, roman immòbil marcant la mateixa direcció constantment (el camp de visió és relatiu a la brúixola i no als objectes). En lloc de rotar el con de visibilitat a mesura que l'usuari es mou, el que es mou l'entorn, els *POIs*.

D'aquesta manera els objectes virtuals –*POIs*- apareixen i desapareixen de la pantalla a mesura que l'usuari canvia de direcció.

A la següent figura (Fig. 4.5) es troba un esquema del funcionament d'aquest bloc. D'una banda apareix la vista de Realitat Augmentada que és la encarregada de gestionar tota la lògica explicada i que sosté tot el pes de la vista.

Quan l'usuari clica sobre un dels *POIs* que apareixen a la pantalla, es carrega la vista de la informació de realitat augmentada en la que es mostra la informació del vídeo –títol, descripció i URL-. Un cop arribat a aquest punt, la última transició que queda és la que duu a l'usuari a reproduir el contingut del vídeo.

Fig. 4.5 Bloc de Realitat Augmentada

Com s'ha comentat amb anterioritat, aquest bloc sencer necessita que al inicialitzar-se se li passi com a paràmetre un vector -*NSArray*- amb tots els punts que s'han de mostrar.

Aquest aspecte és la principal diferencia respecte a la vista dels vídeos sobre el mapa ja que és alhora la seva principal limitació: el fet que es passi el vector de *POIs* directament des de l'inici de l'execució de la vista –al constructor- fa que fins que no re-inicialitzi la vista no es puguin afegir més punts d'interès.

No posseeix cap opció de refresc de les dades així que si, hipotèticament, un usuari A carrega un vídeo a la aplicació mentre l'usuari B té oberta la vista de realitat augmentada, l'usuari A no veurà aquest vídeo fins que surti de la vista i la torni a iniciar.

4.1.3. Vista de Mapes

El bloc de vista de mapes és sens dubte el bloc més complex de tota la aplicació degut al fet que és l'encarregat de gestionar la localització de l'usuari, obtenir tots els vídeos del servidor, mostrar-los sobre un mapa físic mitjançant el *framework* de mapes proporcionat per *Google* i a més poder-los reproduir

quan es seleccionin, a la figura 4.6 s'aprecia la estructura modular que composa aquest bloc.

Fig. 4.6 Bloc de Vista de Mapes

Les tecnologies clau utilitzades en aquest bloc, la majoria d'elles emprades al mòdul Mostrar Vídeos, es resumeixen en:

- *CoreLocation framework*
- *MapKit framework*
- *MediaPlayer framework*
- *QuartzCore framework*
- *XMLParser*

Degut a la complexitat dispar entre els diferents mòduls que composen el bloc, s'exposaran segons el seu nivell de dificultat.

El primer dels blocs a exposar serà el bloc d'informació de vídeo. Aquest bloc només compren una vista –la d'informació de vídeo- i el controlador de la mateixa.

Aquest mòdul té una doble funcionalitat. D'una banda mostra la informació del vídeo seleccionat amb la informació del títol, descripció, autor i coordenades del mateix, i per l'altre banda, estableix la transició –fa de nexe- entre la vista dels vídeos i la reproducció dels mateixos. Segons la catalogació de tipus de vistes, es tracta d'una vista modal.

La seva funció és anàloga a la vista d'informació de vídeo de la capa de realitat augmentada, tot i que diferent pel que fa a la implementació:

Cada cop que es clica una xinxeta del mapa (cada xinxeta representa un vídeo), es crea una instància del controlador del mòdul d'informació de video (passant com a paràmetre el vídeo clicat) i a continuació es mostra la vista amb les dades obtingudes del vídeo passat.

Aquesta vista no té cap element nivell tècnic rellevant ja que pel que fa a programació es tracta d'una classe que mostra en etiquetes -*UILabels*- els valors que rep del constructor al inicialitzar-se i, posteriorment, crea una instància del reproductor de vídeo en cas que l'usuari estigui interessat en el visionat del mateix.

Tota la resta de lògica obeeix a motius de gestió de recursos de memòria: assegurar-se de que quan l'usuari torna a la vista de mapes la instancia s'elimina per estalviar memòria (si no es fes això, cada cop que es cliqués sobre un vídeo es crearia una instància nova) o pel contrari, garantir que la

instancia continua activa si un usuari vol reproduir el vídeo triat, ja que en finalitzar la reproducció del vídeo es torna a la pròpia vista.

A l'Annex C.1 Reproducció de vídeo, es troba la informació sobre el funcionament i la inicialització del framework de reproducció de vídeo així com una explicació de les capacitats del mateix

L'últim bloc que resta comentar de la vista de mapes és, precisament, el que mostra els vídeos sobre els mapes proporcionats per *Google*, s'ha anomenat vista de mapes. Seguint amb la política d'utilitzar el màxim nombre de tecnologies proporcionades per *Apple* als seus *kits* de desenvolupament, es proporciona a continuació una captura de la jerarquia de classes present a aquest bloc generada automàticament amb l'eina de documentació inclosa a *l'XCode*.

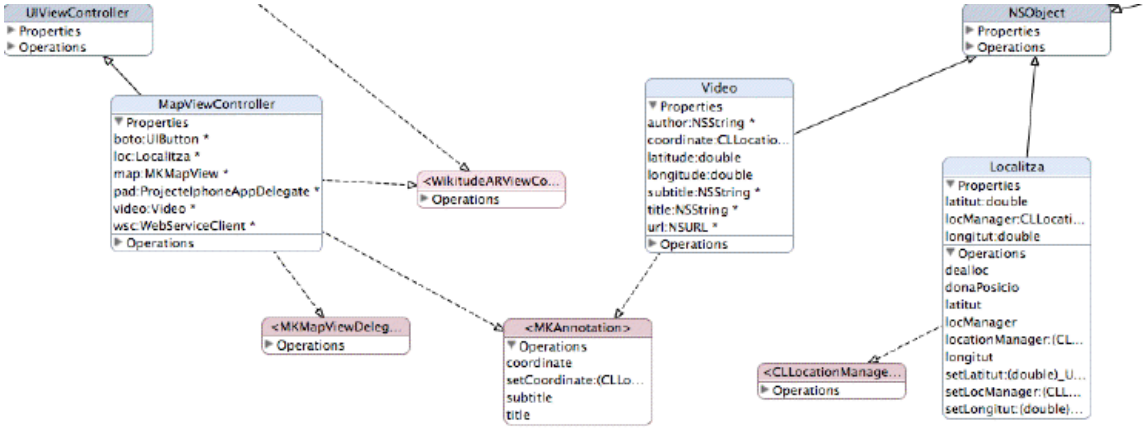

Fig. 4.7 Diagrama d'estructura de classes del mòdul de vista al mapa

En aquest cas s'ha considerat fer ús d'aquest esquema en detriment d'un diagrama UML pel fet que gràficament és més intuïtiu veure quins *frameworks* i tecnologies implementen cadascuna de les classes implicades.

Com s'ha comentat al començament del bloc, la majoria de les tecnologies que empra la aplicació són o bé emprades o bé inicialitzades en aquesta vista així que es procedirà a escrutar-les amb deteniment, però primer és necessari entendre el diagrama.

A la figura 4.7 s'aprecien en color blau les classes que formen part de la vista i en color vermell els delegats que implementen cadascuna de les classes.

Si es mira detingudament el mapa, s'observa que per una banda la classe *MapViewController* i per l'altre les classes Video i Localitza apunten cap a dues classes, *UIViewController* i *NSObject* respectivament.

UIViewController és la meta-classe de la que hereten totes les classes que són controladores de vistes, mentre que *NSObject* és la classe més alta de la jerarquia de classes a *l'Objective-C* equivalent a la classe *Object* del llenguatge Java.

Bé doncs un cop entès l'esquema resulta obvi esbrinar quines tecnologies prenen part d'aquest mòdul simplement mirant les variables –*ivars*- i els delegats que implementa el controlador de la vista *MapViewController*:

D'una banda la classe implementa les variables *loc* i video –variables de tipus Localitza i Vídeo respectivament- i a més a més implementa 3 delegats. Si s'analitzen les dues classes, s'aprecia que de retruc, també implementen un altre delegat i un protocol.

- *WikitudeARViewControllerDelegate*
- *MKMapViewControllerDelegate*
- *MKAnnotationDelegate*
- *CLLocationManager*
- *MKAnnotation Protocol*

D'aquesta informació es pot extreure que aquest bloc implementa els *frameworks* (obviant el *framework* de realitat augmentada propietat de tercers):

- *MapKit framework*
- *CoreLocation framework*

Pel que fa al *MapKit framework*, s'encarrega de gestionar la vista dels mapes proporcionats per *Google*. A més a més s'encarrega de gestionar automàticament els gestos de control multi tàctil com el pessic i zoom per augmentar o disminuir l'escala del mateix i, lògicament, també s'encarrega de gestionar els punts amb informació afegida sobre els mapes, les anotacions o xinxetes.

Per gaudir d'una visió tècnica sobre el framework de mapa i la seva inicialització consultar l'Annex C.2 Visualització de continguts sobre mapa.

L'últim aspecte a cobrir pel mòdul de vista del mapa és la geolocalització. La seva importància és cabdal ja que és una característica transversal a tota la aplicació, tan a l'enregistrament de vídeos com a la visualització dels mateixos.

El framework de localització és el primer que s'executa al iniciar la aplicació, inclús abans de carregar la vista de *login,* hi ha un motiu que justifica aquesta actuació i és la velocitat.

Aquest framework és, potencialment, el més problemàtic ja que és el que més temps requereix per inicialitzar-se correctament; Per al seu correcte funcionament cal que les lectures de posicions que es rebin tinguin un valor de *timestamp* acceptable –totes les posicions que es retornen venen en funció d'un valor que identifica la coordenada amb l'instant de temps en que s'ha pres la mesura.

A més a més, no es garanteix que aquestes lectures vinguin ordenades cronològicament fet que implica que s'hagin de filtrar les lectures obtingudes més antigues.

Pel que fa a la implementació, per solventar aquest problema, s'ha considerat que una posició serà vàlida només si el seu *timestamp* és inferior en valor absolut a 5 segons.

Degut a aquesta limitació, es precisa que aquest objecte sigui inicialitzat quan abans millor per que al moment de pujar un vídeo o localitzar la posició de l'usuari al mapa ja es disposi prèviament d'unes dades de localització vàlides.

Pel que fa a nivell de codi, segueix la mateixa lògica que els anteriors i es resumeix en aquests 3 passos:

- Afegir el delegat a la classe controladora de la vista -en aquest cas la de mostrar vídeos.
- Inicialitzar el component *CLLocationManager*
- Implementar els mètodes del delegat corresponents

De nou, els aspectes relacionats amb la utilització i inicialització del framework CLLocationManager són tractats a l'annex C.3 Geolocalització: Obtenció de les coordenades

Comentar que simplement s'assignen els valors obtinguts a la variable loc de la classe *MapViewController*. A partir d'aquest moment, cada cop que es notifiqui un canvi de posició, la vista del mapa tindrà les coordenades actualitzades.

4.1.4. Pujant Vídeos

L'últim bloc que composa l'aplicació VidGeoLocator és el bloc d'enregistrament i pujada de vídeos al servidor de la aplicació.

En aquest bloc tal i com es mostra a la figura 4.8, es composa de 3 mòduls. El primer mòdul és el de pujar vídeos; com a tal s'encarrega de gestionar quin tipus de video es pujarà al servidor –si és un nou vídeo o per contra és un vídeo enregistrat amb anterioritat al terminal- i executar la pujada mitjançant una mitjançant el mètode *post* del protocol *HTTP*.

Fig. 4.8 Bloc de pujada de vídeos

Les altres dues classes són les implementacions dels dos possibles fonts d'origen de vídeo que s'ha comentat. Per un costat es troba la vista de Càmera pròpia del terminal mitjançant la qual pot enregistrar un vídeo i per l'altre es troba una vista que mostra una llista de tots el vídeos disponibles al terminal.

El primer mòdul a tractar en aquest bloc serà el mòdul de pujada de vídeo al servidor. A la figura 4.9 es pot observar què composa la vista relacionada amb aquest mòdul.

Fig. 4.9 Vista d'enregistrament de vídeos

- D'una banda el primer element que apareix és un commutador que serveix per gestionar el comportament –i el text- del botó grava vídeo.

- A continuació es troben els dos *UITextFields* o cel·les d'entrada de dades per teclat que serveixen per inserir el títol i subtítol pertanyent al vídeo que es disposi a penjar.

- Per últim, es troben els dos botons que donen sentit a tot el mòdul globalment: el botó d'obtenció de vídeos i el botó de pujada del mateix.

Pel que fa a tecnologies, aquest mòdul fa ús de *l'NSXMLParser* i de la instancia de *CLLocationManager* inicialitzada a la vista de visualitzar mapa per obtenir les coordenades actuals..

Tanmateix, degut a la nomenclatura amb que *l'iOS* desa els vídeos enregistrats, quan dos usuaris de dos terminals diferents pugessin un video, es sobreescriurien uns sobre els altres amb un índex de probabilitat molt alt.

És per aquest motiu que s'ha decidit implementar una funció que retorni el *Hash MD5* dels arxius de vídeo i emprar-lo com a nom del propi vídeo.

Al següent esquema es mostra els passos a completar fins a pujar un vídeo al servidor.

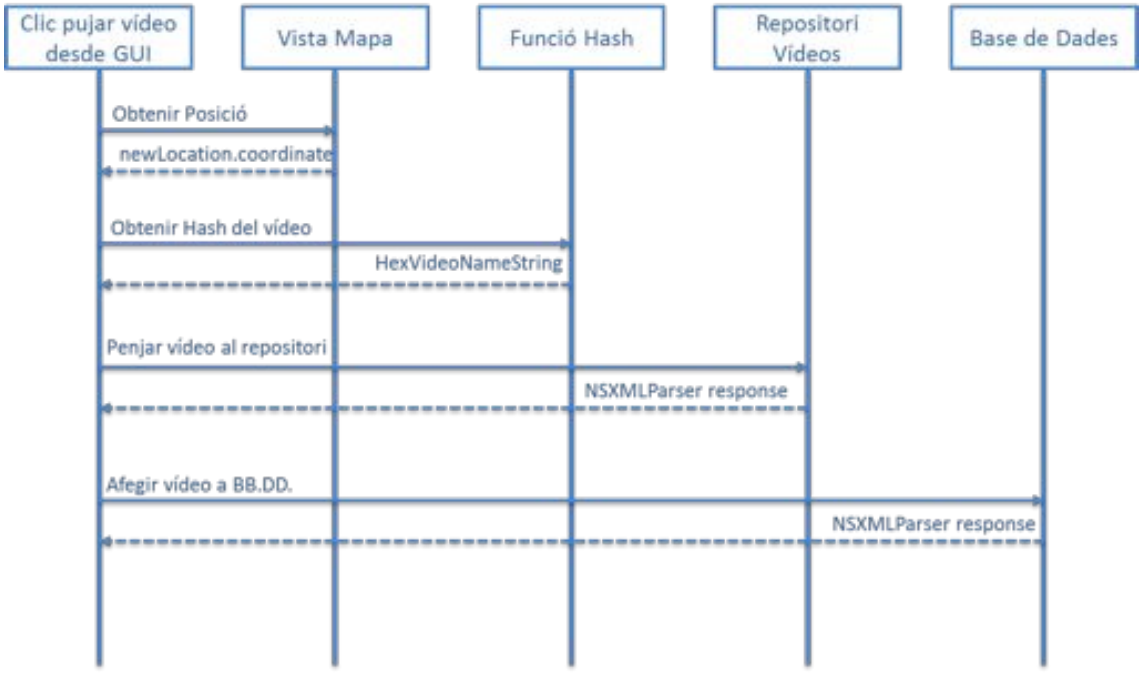

Fig. 4.10 Diagrama seqüencial de pujada de vídeo al servidor

Com es pot veure a la figura 4.10 la pujada d'un vídeo al servidor és un procés de 4 passos.

En primer lloc, un cop es polsa el botó de pujar vídeo a la vista d'enregistrament de vídeos, aquesta accedeix a les dades de posició actual del terminal emmagatzemades a la vista del mapa en la variable loc.

Un cop s'han obtingut les dades de posició, es procedeix a obtenir el resum MD5 generat a partir del fitxer de vídeo. Aquesta funció retorna una cadena en d'una longitud de 32 caràcters que passarà a ser l'identificador del vídeo tant al repositori com a la base de dades.

El tercer pas és realitzar la pujada al servidor. Per fer-ho, es realitza una petició *HTTP post* on s'inclou les dades del vídeo a pujar. Un cop finalitzada la transmissió el servidor retorna el *hash* del fitxer que ha rebut.

Finalment, si els dos *MD5* –el generat prèviament i l'obtingut a la resposta del servidor- coincideixen, s'envia una petició mitjançant el mètode ja explicat de *l'NSXMLParser* al servei web d'actualització de dades.

En aquesta URL s'hi afegeixen els valors obtinguts de:

- Localització
- Usuari
- Títol
- Descripció
- Nom del vídeo

Un cop el procés s'ha completat amb èxit, desapareix el pop up de la pantalla que indica que la tasca està en procés i si l'usuari torna a la vista de mapes i actualitza les dades podrà observar el vídeo acabat d'enregistrar.

El segon mòdul d'aquest bloc és el de selecció de vídeos. La seva funcionalitat és redueix a mostrar una llista amb tots els vídeos que estan enregistrats al telèfon.

```
NSString *  resources = [[NSBundle mainBundle]resourcePath];
NSArray  * aux =
        [ [NSFileManager defaultManager] contentsOfDirectoryAtPath:resources error:nil];

        for (i = 0; i < [aux count]; i++) {
                NSString * obj = [aux objectAtIndex:i];
                if ([obj hasSuffix:@".m4v"] || [obj hasSuffix:@".MOV"]){
                        [vids addObject: obj];
        }
```

Aquesta porció de codi s'encarrega de recórrer tots els arxius que es troben a la carpeta de recursos de la aplicació i observar si el seu sufix és un arxiu de video suportat (en aquest cas, amb extensions *MOV* o *M4V*). En cas afirmatiu, s'afegeixen a un vector que serà passat al constructor de la vista de selecció de vídeos.

Aquesta vista consta únicament d'una llista que s'anirà omplint en funció de la quantitat de vídeos que hi ha a la carpeta de recursos de la aplicació –cal recordar que les aplicacions no tenen visibilitat entre elles, així que un vídeo emmagatzemat a una aplicació A no pot ser trobat per una aplicació B a no ser que estigui a l'àlbum de media del sistema-.

Un cop polsat un vídeo, es mostrarà una alerta com la de la ,que informarà a l'usuari de les seves dues úniques opcions: seleccionar-lo (tornant a la vista de pujar vídeo) o visualitzar-lo (per comprovar que sigui el vídeo que realment vol penjar).

Fig. 4.11 Alerta de notificació a la llista de selecció de vídeos

Per últim, l'últim mòdul del bloc i de la aplicació és el de l'enregistrament de la càmera mitjançant el *framework UIImagePickerController*, el mòdul de càmera de vídeo.

Degut a que aquest mòdul és purament la implementació del framework de gestió de la camèra del dispositiu, s'ha afegit com a annex, (Annex C.4. Utilitzant la càmera del terminal).

CAPÍTOL 5. CONCLUSIONS

En aquest darrer capítol s'exposaran les conclusions a les que s'ha arribat al finalitzar el projecte mitjançant l'anàlisi dels objectius marcats a la fase inicial del mateix, tanmateix es proposaran possibles millores o ampliacions del propi projecte de cara a posteriors revisions. Finalment s'analitzarà el possible impacte mediambiental que suposaria la seva posada en marxa.

5.1. ANÀLISI DELS OBJECTIUS DEL PROJECTE

L'objectiu final d'aquest treball de final de carrera era la realització d'una aplicació per iOS. A més a més, ha servit com a prova de concepte de les capacitats multimèdia i de geolocalització dels dispositius que funcionen sota aquest sistema operatiu.

Sobretot ha imperat una màxima alhora de realitzar el projecte i ha estat el fet d'utilitzar el màxim nombre de tecnologies que proporciona *Apple* i el seu *SDK* per a *l'iOS* tot i que en algun cas no hagi estat la solució òptima o més simple, ja que, un altre cop, el principal repte residia en dominar i comprendre la plataforma.

Per poder arribar a dominar la plataforma iOS, a la introducció del projecte s'havien marcat una sèrie d'objectius, a continuació es valorarà la consecució de cadascun d'ells.

- Estudi de l'ecosistema mòbil d'*Apple*:

S'ha realitzat una introducció completa de la plataforma que tracta des de la creació de la mateixa fins al model de negoci que s'hi ha establert al voltant, sense descuidar els models de dispositius que el composen ni comparatives amb els principals rivals del sector.

- Estudi del llenguatge de desenvolupament per iOS.

S'ha realitzat un estudi a nivell tècnic dels components que composen el sistema operatiu iOS. A més a més s'han explicat les principals tecnologies pertanyents a aquests components a nivell teòric i pràctic.

S'ha introduït el llenguatge Objective-C, necessari per tal de poder desenvolupar per l'iOS. S'han desenvolupat els conceptes i patrons de funcionament fonamentals pel desenvolupament d'aplicacions en aquest llenguatge. Tanmateix també s'ha introduït exemples pràctics que demostren l'aplicació real dels mateixos.

- Estudi de tecnologies incloses al dispositiu *iPhone*.

S'ha identificat, explicat i comparat amb els dispositius de la competència les diferents tecnologies incloses a l'*iPhone*, tals com l'acceleròmetre, la brúixola, la càmera de fotos, etc. A més s'ha explicat el funcionament dels *frameworks* necessaris per al desenvolupament d'aplicacions que en facin ús.

- Estudi sobre la realitat augmentada.

S'ha presentat el concepte de la realitat augmentada. Incloent una explicació del funcionament de la tecnologia. S'ha utilitzat una API per afegir aquesta funcionalitat a la aplicació desenvolupada.

S'han explicat el rol que juguen les tecnologies necessàries per el desenvolupament d'una aplicació de realitat augmentada, concretament l'acceleròmetre, la brúixola i el receptor GPS.

- Disseny d'una aplicació pel sistema operatiu *iOS*.

S'han introduït els conceptes que estan involucrats en el desenvolupament d'una aplicació per iOS, concretament per *iPhone*; per fer-ho, s'ha explicat el model de desenvolupament MVC, recomanat per *Apple*. A més a més s'ha exposat quin és el cicle de vida d'una aplicació d'acord a l'última versió del sistema operatiu –incloent la multitasca-.

Tanmateix s'ha parlat de les fases que implica el desenvolupament d'una aplicació –d'una banda el codi i de l'altre la interfície gràfica- i cóm s'uneixen mitjançant la utilització d'etiquetes –IBAction i IBOutlet- que relacionen ambdues fases.

- Implementació d'una aplicació per *iPhone*.

Finalment la part amb més càrrega de feina del projecte s'ha invertit en el disseny i desenvolupament d'una aplicació per a l'*iPhone*, VidGeoLocator, que aglutina tots els conceptes exposats durant el treball.

Un cop finalitzat el desenvolupament de la aplicació, concretament el client per iOS, s'ha obtingut un software molt modular i fàcilment adaptable a diferents entorns .

Un dels camps que s'ha mostrat interessat en aquest tipus d'aplicació és el camp de e-salut i e-dependència. Concretament la fundació i2Cat s'ha mostrat interessada en les possibilitats de l'aplicació en el camp de la e-salut. Tan és així que, finalment, s'ha engegat un projecte, que actualment està en fase de proves, basat en aquest projecte.

El projecte s'anomena eHelp i consisteix en un sistema d'ajut en cas d'emergència on tan sols amb un clic, l'usuari pot enregistrar un vídeo explicant el problema que té –en cas de ferida gràcies al vídeo es pot fer un primer anàlisi del grau de perillositat- i enviar-lo a un metge que el rebrà

immediatament i, mitjançant la geolocalització, podrà o bé enviar ajuda de forma més rapida o bé dirigir al usuari cap al centre més proper per tractar-lo.

5.2. PROPOSTES DE MILLORA

Un dels primers aspectes amb què es notaria una substancial millora de la aplicació seria la implementació pròpia de la vista de realitat augmentada. La utilització d'una API externa inestable quan no s'executa a l'exterior és sens dubte el taló d'Aquil·les de la aplicació.

D'altra banda s'hauria d'afegir un mecanisme de cerca de vídeo per títol, descripció o usuari. Tot i que aquest últim està implementat al servidor, no s'ha afegit finalment al client. És una característica gairebé imprescindible a mesura que la aplicació vagi augmentant el nombre d'usuaris i per tant el nombre de vídeos es multipliqui.

Pel que fa a seguretat, s'hauria d'afegir xifrat *SSL* a les comunicacions per protegir les comunicacions entre client i servidor.

Finalment es podrien implementar les notificacions *push* per informar a un usuari A quan un usuari B reprodueixi un dels vídeos pujats pel primer.

5.3. ANÀLISI DE L'IMPACTE MEDIAMBIENTAL

Un dels punts destacats al projecte ha estat l'ús dels nous dispositius amb un alt nivell de mobilitat. A més del factor de la mobilitat, també entra en joc el factor de l'estalvi energètic: dispositius que s'equiparen pel que fa a prestacions als ordinadors de sobretaula però que tenen un consum molt menor proporcionen un estalvi energètic avui dia que ni pot ni ha de passar desapercebut a ningú.

Amb el desenvolupament de l'aplicació VidGeoLocator, es permet a l'usuari dur a terme operacions com la visualització o enregistrament de vídeos directament sobre el terminal, evitant així haver de realitzar aquestes operacions des d'un ordinador, estalviant conseqüentment l'alt consum d'aquests últims.

BIBLIOGRAFIA I REFERÈNCIES

BIBLIOGRAFIA

Knaster, S., Dalrymple, M., *Learn Objective-C on the Mac,* Apress, New York, Gener 2009.

Mark, D., Nutting, J., LaMarche, J., *Beginning iPhone 4 development: Exploring iOS SDK*, Apress, New York, 2011.

Mark, D., Nutting, J., LaMarche, J., *Learn Cocoa on the Mac*, Apress, New York, 2010

Vila, L., *Moving, una aplicación para Android***,** TFC, Universitat Politècnica de Catalunya (UPC), Octubre 2010.

REFERÈNCIES

[1] Pàgina principal del projecte open source Mach Kernel utilitzat tant a iOS com a OS X. (Recurs en línia). [Última Consulta: 29 de març de 2011]. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html.

[2] Explicació de la diferència entre directives de pre-processat utilitzades a Objective-C. (Recurs en línia). [Última Consulta: 27 de març de 2010]. https://developer.*Apple*.com/library/ios/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocDefiningClasses.html#//*Apple*_ref/doc/uid/TP30001163-CH12-TPXREF123.

[3] Explicació del funcionament dels protocols a Objective-C. (Recurs en línia). [Última Consulta: 3 d'abril de 2011]. https://developer.*Apple*.com/library/ios/#documentation/General/Conceptual/DevPedia-CocoaCore/Protocol.html#//*Apple*_ref/doc/uid/TP40008195-CH45-SW3.

[4] Explicació del patró del delegat. (Recurs en línia). [Última Consulta: 3 d'abril de 2011]. https://developer.*Apple*.com/library/ios/#documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html.

[5] Explicació i definició dels WebServices de tipus RESTful.(Recurs en línea) [Última consulta 15 Abril de 2011]. http://www.oracle.com/technetwork/articles/javase/index-137171.html

[6] Explicació de l'algoritme MD5 que s'aplica als vídeos al penjar-los al servidor (recurs en línea) [Última consulta 1 de Maig de 2011]. http://es.wikipedia.org/wiki/MD5

[7] Informació API realitat augmentada Wikitude. (Recurs en línia). [última consulta 8 de Febrer de 2011]. http://www.wikitude.org/

[8] Tenda online d'aplicacions alternativa a la App Store de Mac només accessible per a dispositius liberalitzats. (Recurs en línia). [Última Consulta: 20 de març de 2011].
http://cydia.saurik.com/store/.

GLOSSARI D'ACRÒNIMS, DEFINICIONS I SIGLES

Còdec - És un esquema que regula una sèrie de transformacions en un senyal o informació. Els còdecs tant poden transformar un senyal a una forma codificada (usada per la transmissió o encriptació) com fer-ho al revés, donar la senyal adequada per a la seva visualització o edició (no necessàriament la forma original) a partir de la forma codificada.

Framework - Estructura conceptual i tecnològica de suport definida, normalment amb mòduls de software concrets, amb base a la qual un altre projecte de software pot ser organitzat i desenvolupat. Típicament pot incloure suport de programes, biblioteques i un llenguatge interpretat entre altres programes per ajudar a desenvolupar i unir els diferents components d'un projecte.

Gpu (*Graphic Process Unit*) **-** dispositiu dedicat a la generació de gràfics per a ordinadors personals, estacions de treballs o consoles de videojocs.

Hibernate - és una framework pel mapeig d'objectes relacionals (ORM) per aplicacions Java, sobre una base de dades relacional

HTTP (*Hypertext Transfer Protocol*) **-** Protocol de la capa d'aplicació utilitzat a cada transacció de la World Wide Web.

iTunes - és un reproductor multimèdia desenvolupat per *Apple* que permet reproduir, organitzar i comprar música i vídeos digitals. El programa és alhora una interfície per a gestionar els continguts dels populars reproductors iPod, *iPad* i *iPhone*.

Kernel - El Kernel o nucli és el component central de la majoria de sistemes operatius, fa de nexe entre les aplicacions i el processament de dades reals fet a nivell hardware.

MMS (*Multimedia Messaging System*) **-** és un estàndard de missatgeria que permet als telèfons mòbils enviar i rebre continguts multimèdia, incorporant so, vídeo, fotos o qualsevol altre contingut disponible en el futur.

MySQL - És un sistema de gestió de bases de dades relacional multi-fil i multiusuari, que usa el llenguatge SQL.

REST - (*Representational State Transfer*) **-** Tècnica d'arquitectura software per a sistemes hipermèdia distribuïts com la xarxa internet.

SDK (*Software Development Kit*) **-** És generalment un conjunt d'eines de desenvolupament que permeten a un programador crear aplicacions per a un sistema concret.

Servlet - Objectes que funcionen dins del context d'un contenidor de servlet o aplicacions. La seva funció és generar pàgines web de forma dinàmica a partir dels paràmetres de la petició que envií el navegador web.

SQL - És un llenguatge estàndard de comunicació amb bases de dades relacionals. És a dir, un llenguatge normalitzat que permet treballar amb la majoria de bases de dades relacionals.

UIButton – Components de la interfície gràfica de l'iOS. Aquest component és representat per un botó a la interfície que generalment té associat un o més mètodes que s'executaran quan sigui premut.

UITabBarController – Component de la interfície gràfica de l'iOS. Aquest component és un controlador que permet separar dues o més vistes en pestanyes per facilitar la visualització de continguts independents**.**

UITextField - Component de la interfície gràfica de l' iOS. Aquest component serveix de mètode d'entrada de dades per part de l'usuari mitjançant la invocació d'un teclat virtual a la pantalla del dispositiu.

WAP (*Wireless Application*) **-** És un estàndard obert internacional per a aplicacions que utilitzen les comunicacions sense fil. s tracta de l'especificació d'un entorn d'aplicació i de conjunt de protocols de comunicacions per a normalitzar el mode en què els dispositius sense fil, es poden utilitzar per a accedir al correu electrònic, grup de notícies i altres.

XML *(eXtensible Markup Language)* **-** Metallenguatge d'us general que serveix per a definir altres llenguatges de programació o formats d'intercanvi d'informació segons diverses necessitats.

ANNEXES

A. iPhone: EL TERMINAL MÒBIL D'APPLE

A.1. INTRODUCCIÓ

La proposta d'aquest projecte de realitzar una aplicació per a iOS va lligada específicament a un terminal sobre el que es desenvoluparà: l'*iPhone*.

En aquest capítol es contextualitzarà el dispositiu dins del mercat actual i a més a més es durà a terme un repàs a les alternatives similars al mateix que es poden trobar avui dia al mercat i, tanmateix, es farà una comparativa entre les característiques de cadascun d'ells.

Finalment el capítol prosseguirà amb una reflexió sobre quins avantatges i quines motivacions porten a un desenvolupador a interessar-se per la programació per *iOS*.

A.1.1 Història i evolució

Avui dia al sector mòbil del mercat de les telecomunicacions hi ha una grandíssima diversitat d'oferta de terminals amb capacitats molt properes als ordinadors de sobretaula. Si bé no fa tants anys el concepte de terminal potent, era un terminal que era capaç de fer fotos, reproduir cançons en mp3 i disposar d'una pantalla amb 65.000 colors, avui dia s'ha passat a valorar molt més l'arquitectura interna del propi terminal (processador/s, *gpu*, connectivitat a internet, multitasca).

Bona culpa d'aquesta transició la té el terminal que ens ocupa. Abans de l'arribada de l'*iPhone* al 2007, al mercat no dedicat (el que no era de negocis) estava gairebé monopolitzat pels terminals de la finlandesa *Nokia* que corrien o bé sobre *Symbian* o bé sobre *Windows Phone*.

En aquell moment, tot i que començaven a aparèixer terminals com el *Nokia* N95 que incorporaven *WiFi*, ningú es plantejava realment la possibilitat de navegar per internet amb el telèfon mòbil i la experiència d'usuari era força precària.

Aquest va ser un dels canvis més importants que va introduir l'*iPhone*. Tot i que la primera versió comptava amb handicaps significatius respecte als seus competidors (no podia enviar *MMS*, fer videotrucades ni enregistrar vídeo i a és a més no tenia connectivitat 3G) va ser un èxit rotund. Tan va ser així que la indústria sencera va començar a desenvolupar els nous productes centrant-se en aquest terminal.

Es va abandonar la tecnologia *WAP*, un protocol que mai va acabar d'arrencar i es va començar a implantar la idea de navegadors web estàndard per als dispositius mòbils. A aquest fet cal afegir-hi la explosió de les interfícies tàctils

acompanyat del canvi de factor de forma de les pantalles que acompanyaven als dispositius, a la figura A.1 s'aprecia aquest canvi.

Fig. A.1 Comparació entre el disseny de terminals mòbils i l'iPhone

Si es mira detingudament a aquests terminals –sobretot els 2 primers- i es compara amb el disseny del terminal d'*Apple*, es veuen clarament similituds palpables, des de la ubicació dels botons fins a la proporció i tamany de la pantalla

Per altra banda, i deixant de banda la competència, si es duu a terme una anàlisi introspectiva al propi terminal queda palesa una evolució dintre del propi dispositiu.

Des del primer terminal llençat al mercat al 2007 fins al terminal actual, l'*iPhone* 4, hi ha 4 versions del mateix terminal. Algunes versions només implicaven un canvi a nivell a nivell de hardware mentre que altres a més a més hi afegien un redisseny del terminal.

A la següent taula, compararem les versions dels terminals per despres parlar de les diferencies més rellevants:

Taula A.1 Comparativa del hardware del les versions de l'*iPhone*

	iPhone	*iPhone* 3G	*iPhone* 3GS	*iPhone* 4
Processador	ARM11 412Mhz	ARM11 412Mhz	ARM Cortex A8 620Mhz	ARM Cortex A4 800Mhz
Pantalla	320x480 163 ppp	320x480 163 ppp	320x480 163 ppp	960 x 640 326 ppx
Càmera	2 Mpx	2 Mpx	3 Mpx	5 Mpx
Memòria	128 MB	128 MB	128 MB	512 MB
Acceleròmetre	✓	✓	✓	✓
GPS	✗	✓	✓	✓
3G	✗	✓	✓	✓
Brúixola	✗	✗	✓	✓
Giroscopi	✗	✗	✗	✓

El primer *iPhone*, com s'ha comentat a la introducció, tot i resultar trencador, no estava a l'avantguarda dels terminals més punters del moment pel que a especificacions es refereix. Comparativament, el primer *iPhone* i la primera revisió del mateix, com bé indica el seu nom es caracteritza per la incorporació de la tecnologia 3G al terminal juntament amb una altra petició molt demandada, el GPS.

Si es dóna una ullada a la taula, s'observa que comparteixen la resta d'especificacions. Juntament amb aquest redisseny, va aparèixer l'iOS3 carregat amb optimitzacions software i millores molt demandades com per exemple la funció retalla i enganxa o la ja comentada inauguració de la AppStore.

La següent versió va ser una petita revolució a nivell de hardware dins del mateix terminal. El canvi va ser positiu en tots els aspectes, més potencia pel processador, més qualitat a la càmera de fotos –permetent la gravació de video no possible fins aquell moment- i un nou sensor, la brúixola que va obrir la porta a les aplicacions de realitat augmentada.

Finalment, la quarta i actual versió del terminal, l'*iPhone* 4 torna a representar un pas endavant pel que fa a especificacions hardware i de disseny.

En el primer aspecte, d'una banda es potencia encara més el processador amb una freqüència de rellotge limitada a 800Mhz –tot i que la seva freqüència stock es d' 1Ghz- per minimitzar el consum de la bateria i la novedosa retina display amb una resolució 4 cops superior al seu predecessor. A més a més, s'afegeix un giroscopi al terminal per dotar-lo d'una precisió més acurada per les aplicacions que així ho requereixin.

En el segon aspecte, es redissenya completament el terminal (no com a les altres versions on el redisseny venia donat per la reducció de tamany) deixant enrere la forma semi-ovalada de la carcassa anterior característica de la casa i passant a un disseny completament llis mostrat a la figura a peu de pàgina (Fig. A.2).

Fig. A.2 Comparativa redisseny iPhone 4 i 3GS

A.1.2. Per què desenvolupar per *iPhone*?

L'èxit de l'*iPhone* per damunt del sistema operatiu és que ha aconseguit engrescar a una massa de desenvolupadors prou gran com per haver realitzat més de 400.000 aplicacions en 4 anys aproximadament.

De cara al desenvolupador, la plataforma ofereix motius prou atractius com per llençar-se a aprendre-la.

Per una banda com és lògic, està el factor econòmic. *Apple* cedeix el 66% dels guanys generats per cada aplicació als creadors de les mateixes i permet lliurement fixar el preu de cada aplicació.

Tot i que no és la tònica general, es cert que gràcies a aquest model de negoci, algunes companyies com per exemple *Rovio* i el seu famós *Angry Birds* han aconseguit ingressos milionaris gràcies a aquesta política de vendes.

Un altre factor molt important és la pirateria. Avui dia és un dels aspectes més preocupants del sector. Si un estudi desenvolupa una aplicació, a les poques hores de llençar-la al mercat estarà replicada a servidors de descàrrega directa o indexada als *trackers* de P2P.

Per lluitar contra aquest fenomen, apareix la *AppStore* com l'únic mitjà pel qual un usuari es pot descarregar les aplicacions de l'*iPhone* ja siguin gratuïtes o de pagament. D'aquesta forma, centralitzant única i exclusivament les aplicacions en un sol mitjà de distribució es pot garantir que quan un usuari vulgui una aplicació haurà de passar prèviament per caixa[4].

A.2 CARACTERÍSTIQUES DEL TERMINAL

Pel que fa a nivell hardware, es comentaran les característiques principals dels terminals com puguin ser processador, memòria RAM i capacitats multimèdia, mentre que pel que fa a l'apartat de software la comparació serà a nivell de sistemes operatius que regenten els terminals.

A continuació es presenten molt breument els candidats amb les seves plataformes corresponents[5]

[4] Existeixen alternatives no oficials a la AppStore com Cydia [8] que requereixen modificar el firmware del terminal per a que funcionin. Aquestes proporcionen un altre repositori d'on baixar-se aplicacions, principalment aquelles rebutjades per Apple mitjançant la seva política d'admisió.

[5] La plataforma Windows Phone 7 no ha estat inclosa tot i haver estat presa en consideració degut a la seva fase de maduresa. En un futur, i sobretot ara que s'acaba de forjar la col·laboració entre la finlandesa Nokia i Microsoft, pot arribar a ser un dels més seriosos competidors del terminal d'Apple.

A.2.1. Alternatives al mercat

A l'actualitat, tot i tenir un mercat força polaritzat, trobem 4 sistemes operatius que gairebé monopolitzen el mercat de la telefonia mòbil: *Blackberry*, *Android*, *Symbian* i *iOS*.

Cadascun té un terminal abanderat de la marca i el que farem a aquest apartat serà dur a terme una comparativa amb l'*iPhone*, però primer presentem als candidats.

Per part de *Blacberry* de la Canadenca *Research* In *Motion*, té una gran presencia en entorns empresarials. El terminal que compararem amb l'*iPhone* serà la *Blackberry Torch*.

Per part d' *Android* del gegant *Google*, té cada cop una presència més gran al mercat degut a la seva política *Open Source* fet que fa que moltes companyies optin per aquest sistema operatiu a cost gairebé 0. El terminal triat en aquest cas serà el *Google Nexus One* de la taiwanesa *HTC* degut a la proximitat temporal en que van ser presentats.

Finalment, per tancar aquesta comparació, es recorrerà al *Nokia N8* que munta un sistema operatiu *Symbian*, sens dubte el sistema operatiu més estès abans de l'arribada dels *smartphones*.

La comparació serà a dos nivells, primer a nivell Hardware del terminal, on es comparen les característiques entre ells posteriorment es realitzarà una comparació a nivell de software on es comparen els diferents sistemes operatius.

A.2.2. Característiques Hardware

De les dades de la taula 5 es poden extreure algunes conclusions interessants. La primera de totes és que, a priori, pel que fa a processadors hi ha dos estrats prou clars. El primer estrat, trobem l'*iPhone*4 i el *Nexus* One, amb un processador que pràcticament duplica en cicles per segon als altres dos competidors.

Taula A.2 Comparativa entre telèfons intel·ligents

	iPhone 4	Nexus One	Nokia N8	Blackberry Torch
Sistema Operatiu	*iOS*	*Android 2.1*	*Symbian^3*	*Blacberry OS 6.0*
Processador	*Apple A4 1Ghz*	*1Ghz Qualcomm Snapdragon*	*ARM11 680 Mhz*	*624 Mhz*
Memòria RAM	*512 MB*	*512 MB*	*256 MB*	*512 MB*
Pantalla	*3,5" Retina Display (326 ppi)*	*AMOLED*	*3,5" OLED*	*3,2" TFT*
Acceleròmetre	✓	✓	✓	✗
Giroscopi	✓	✗	✗	✗
Multi tàctil	✓	✓	✗	✓
GPS	✓	✓	✓	✓
Càmera frontal	✓	✗	✓	✗
Càmera	*5 MP*	*5 MP*	*12 MP*	*5 MP*

Aquest duel se l'emporta el *Nexus One* –recordem que l'*iPhone* treballa per sota de la seva freqüència *stock*-, ara bé, si s'analitza amb deteniment, s'arriba a la conclusió que la freqüència del processador no té per que ser un factor determinant.

Realment cal mirar quant optimitzat està un sistema. Aquest punt serà motiu de debat al següent apartat però pel que fa a nivell hardware, a més potencia més consum i per tant menys duració de la bateria. Si per aconseguir la mateixa experiència d'usuari (o similar) cal que un terminal gairebé dobli a un altre en velocitat de rellotge, clarament el guanyador seria el terminal més optimitzat.

Analitzant una mica més les característiques s'aprecia com els tamany de la pantalla ronden les 3.5 polzades, tot i això, si que hi ha una divergència entre tots els sistemes pel que fa a la resolució i tipus de pantalla.

Mentre que l'*iPhone* es desmarca superpoblant la superfície de la pantalla de píxels amb la seva innovadora Retina Display, el *Nexus One* i *l'N8* trien la tecnologia AMOLED i OLED respectivament rellevant a l'últim lloc a la *Blackberry Torch* i la seva pantalla amb tecnologia TFT.

Pel que fa a sensors que acompanyen a dispositiu *l'Phone* és qui més varietat té gràcies a la incorporació del giroscopi. Per contra destaca molt per damunt

de la resta la càmera fotogràfica del *Nokia N8* considerada actualment la millor *camèra* fotogràfica al mercat de la telefonia mòbil.

Com s'ha pogut comprovar tots els dispositius tenen una base força similar pel que fa a hardware, amb unes característiques comuns: GPS, Càmera fotogràfica, pantalla tàctil....

B. HIBERNATE ANNOTATIONS

La classe analitzada en aquest cas és la declaració de la classe usuari. Com s'aprecia, segueix el patró *POJO* però a més a més s'hi ha declarat, en certs casos –als *getters*-, unes propietats que comencen amb *@*.

Aquestes propietats, s'anomenen *annotations* i és el que *hibernate* utilitza per *mapejar* els valors dels atributs a la base de dades prescindint d'arxius XML per cada classe a *mapejar* definint els atributs. D'aquesta forma s'independitza i relaciona la declaració dels camps de la base de dades de la declaració dels mètodes de la classe.

Mitjançant aquesta tècnica es converteix un objecte, la classe, en una entitat relacional de la base de dades.

```
@Entity
@Table(name = "Usuaris")
public class User {
	protected String name;
	protected String mail;
	protected String password;

	public User() {
		super();
	}
	public User(String name, String mail, String password) {
		super();
		this.name = name;
		this.mail = mail;
		this.password = password;
	}
	@Id
	@Column(name = "name", nullable = false, unique = true)
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	@Column(name = "mail", nullable = false, unique = false)
	public String getMail() {
		return mail;
	}
	public void setMail(String mail) {
		this.mail = mail;
	}
	@Column(name = "password", nullable = true, unique = false)
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
}
```

C. APIS I FRAMEWORKS EXISTENTS A IOS

C.1. REPRODUCCIÓ DE VIDEO

Pel que fa a la vista de reproducció multimèdia, *Apple* ha fet una molt bona feina:

```
- (IBAction) playVideo{

MPMoviePlayerViewController *player = [[MPMoviePlayerViewController alloc]
initWithContentURL: video.url];
        [player setModalTransitionStyle:UIModalTransitionStyleCrossDissolve];
        [self presentModalViewController:player animated:YES];
        player.moviePlayer.movieSourceType   = MPMovieSourceTypeFile;
        [player.moviePlayer play];
        [player release];

}
```

Aquest és tot el codi necessari per reproduir un vídeo gràcies al *MediaPlayer framework*. A continuació s'exposa el funcionament del mateix:

En primera moment es crea una instancia d'un controlador de vista de tipus *MPMoviePlayer*. Aquest controlador gestiona la vista del reproductor multimèdia per defecte de *l'iOS* i és exactament la mateixa interfície que tot usuari d'un *iPhone* o un iPod Touch ha vist quan reprodueix un vídeo o una pel·lícula obtinguda *d'iTunes*.

Tornant a la instancia, en aquest cas es crea una instancia de la classe *MPMoviePlayerViewController* amb el constructor *initWithContentURL*. Òbviament, el paràmetre que correspon al constructor és la pròpia direcció URL del video allotjat al servidor de la aplicació.

Ja està. Aquesta és tota la feina que cal per configurar el reproductor de vídeo. A continuació segueixen tot un conjunt de sentencies que s'encarreguen de controlar com es carregarà la vista i quin tipus de fitxer ha de gestionar .

D'una banda la sentència [*Player setModalTransitionStyle*], s'encarrega de controlar quin efecte de transició –canvi entre la vista anterior i la vista de reproducció- es realitzarà. En aquest cas es tracta d'un efecte d'atenuació.

La següent crida s'encarrega de mostrar la vista. A destacar el fet que es carrega com si es tractés d'una vista modal ja que el funcionament de la aplicació ha de permetre retrocedir a la vista d'informació d'usuaris de forma ràpida.

Per aconseguir tal propòsit s'afegeix el reproductor com una vista just al capdamunt de la vista d'informació de vídeo i s'aconsegueix que quan es descarregui la vista de reproducció es mostri directament la vista amb la informació del vídeo.

Les dues últimes sentències rellevants a la vista són de control. Una de control de contingut i l'altre de control de reproducció.

La primera s'encarrega d'indicar quin tipus de Media és (si es tracta de la reproducció d'un fitxer d'una mida finita o si pel contrari es reprodueix d'una font de mida desconeguda com pugui ser una emissora de radio per internet).

La segona és obvia, s'encarrega d'engegar la reproducció del contingut del vídeo.

Finalment, la última sentencia s'encarrega d'eliminar la instancia del controlador per tal que cada cop que es vulgui reproduir un video no es creïn múltiples instancies.

Recuperant el fil sobre els controls de la vista, en aquest cas dels controls de reproducció, cal dir que s'afegeixen per defecte al carregar la vista. No només això, a més a més, la vista gestiona automàticament la reproducció del vídeo en horitzontal o vertical en funció de la posició en que l'usuari sosté el dispositiu mitjançant l'acceleròmetre com s'aprecia a la figura C.1.

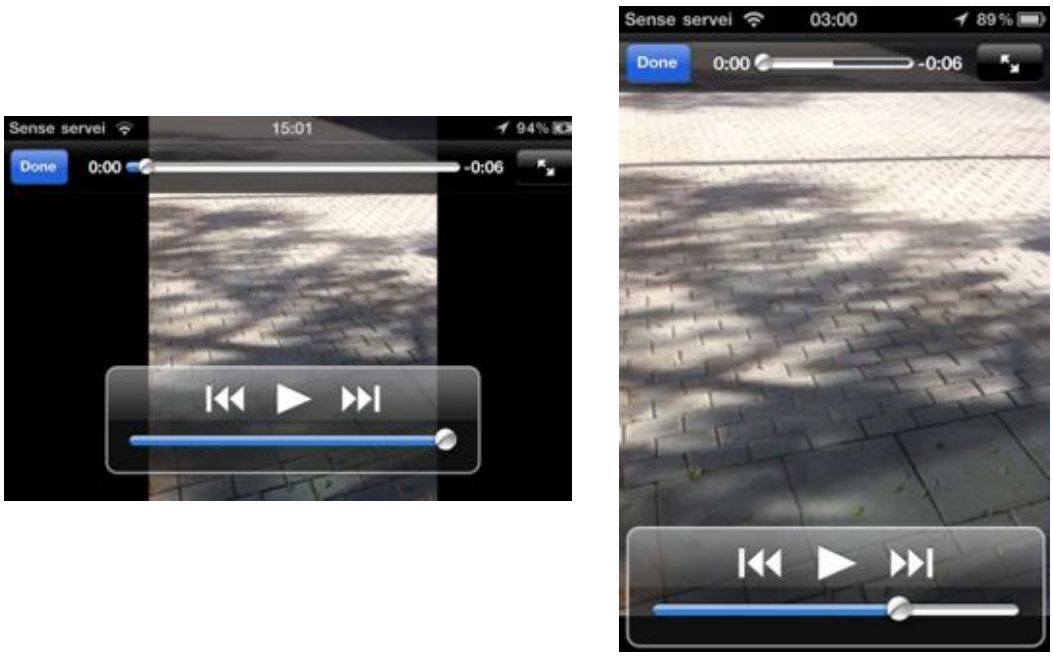

Fig. C.1 Reproducció en funció de la posició del terminal

C.2. VISUALITZACIO DE CONTINGUTS SOBRE MAPA

Per inicialitzar un mapa calen les següents crides:

```
map.delegate = self;
map.mapType=MKMapTypeHybrid;
MKCoordinateRegion regio;
        regio.center.latitude       = [loc latitut];
        regio.center.longitude      = [loc longitut];
        regio.span.latitudeDelta    = 0.2f;
        regio.span.longitudeDelta   = 0.2f;
[map setRegion:regio animated:YES];
[map addAnnotations:[wsc videos]];
map.showsUserLocation = YES;
```

En primera instancia es declara el propi objecte mapa (instancia de *MKMapView*) com a receptor dels esdeveniments que generi asíncronament la vista del mapa com pugui ser per exemple una notificació que l'usuari ha realitzat un gest *multitàctil* per ampliar una determinada regió del mapa o bé que un usuari ha clicat un element del propi mapa.

Realitzant aquesta assignació fem que tota la lògica de la classe quedi dins de la mateixa i per tant, s'hi hauran d'afegir la implementació dels mètodes desitjats del delegat.

A continuació es defineix quin tipus de mapa es vol mostrar. Es pot triar igual que al servei de *Google* per als ordinadors de sobretaula entre la vista satèl·lit, la vista de mapes política, i la vista mixta que aglutina la vista satèl·lit amb la informació afegida per la vista de mapes – nom dels carrers, carreteres, *etc*-.

La següent porció de codi serveix per centrar la vista del mapa sobre una regió. Aquesta regió –la classe *MKCoordinateRegion*- es genera creant un paral·lelogram al voltant d'un punt central. A més a més, el paral·lelogram es conforma amb un valor d'amplada tant de latitud com de longitud que definiran la zona.

D'aquesta manera, contràriament al que pugui semblar, un valor d'amplada superior definirà una vista més allunyada ja que amb la mateixa vista s'ha de mostrar una zona més gran.

Pel que fa al punt central del mapa, s'ha decidit que sigui el lloc on es troba l'usuari –les seves pròpies coordenades de latitud i longitud obtingudes mitjançant geolocalització- per tal que un cop es carreguin els vídeos, trobi aquells que estan geogràficament més a prop de la seva àrea i puguin ser potencialment més útils.

Finalment, cal afegir els vídeos amb les anomenades *MKAnnotations*, les xinxetes o banderoles que indiquen llocs d'interès superposats al mapa.

MKAnnotation és un protocol, com hem definit abans, a grans trets un protocol proveeix a una classe d'una sèrie de mètodes que obligatòriament han de constar implícitament a la implementació de dita classe.

En aquest cas concret, que una classe implementi el *MKAnnotation protocol* significa, en termes d'afegir codi, que aquesta classe tingui 3 atributs –*title*, *subtitle* i *coordinate*- i els seus corresponents *getters* i *setters*.

A VidGeoLocator, és lògic que la única classe que hagi d'implementar aquest protocol sigui la classe vídeo ja que aquesta és la única informació que per una banda esta geolocalitzada i per l'altra es vol mostrar sobre el mapa.

S'ha explicat cóm es crea i inicialitza un mapa i per l'altra com es creen les banderes que hi apareixen però falta el pas que enllaça les dues parts: Què s'ha de fer per mostrar aquestes últimes sobre el propi mapa?

```
- (MKAnnotationView *) mapView:(MKMapView *)mapView viewForAnnotation:(id
<MKAnnotation>) annotation
{
        MKPinAnnotationView *annView =nil;
        annView = [[MKPinAnnotationView alloc] initWithAnnotation:annotation reuseIdentifier:
@"currentloc"];
                annView.pinColor        = MKPinAnnotationColorGreen;
                annView.animatesDrop = TRUE;
 return annView;
}
```

En aquesta porció de codi trobem la resposta. Aquest és un dels mètodes proporcionats pel delegat *MKMapViewDelegate* i és cridat cada cop que es genera un canvi en el mapa i s'ha de col·locar una anotació sobre el mateix.

En aquest cas, un exemple molt simplificat que consisteix en la creació d'una anotació en forma de xinxeta a la que posteriorment s'assigna el color verd –es poden triar només tres colors; verd, vermell i violeta-, s'indica que quan aparegui per primer cop ho faci amb una animació i finalment es retorna la instancia perquè es visualitzi. Un cop més, la feina que resta al desenvolupador és força reduïda.

C.3 GEOLOCALITZACIÓ: OBTENCIÓ DE LES COORDENADES

La porció del codi que realitza aquests tres passos segueix a continuació:

```
loc = [[CLLocationManager alloc] init];
        loc.delegate  = self;
        loc.distanceFilter = kCLDistanceFilterNone;
        loc.desiredAccuracy = kCLLocationAccuracyBest;
[loc startUpdatingLocation];
```

En aquesta secció de codi es mostra la inicialització de la classe *CLLocationManager*. S'aprecia la creació de la instancia, la assignació del delegat a la pròpia classe que l'ha instanciat i per últim s'aprecien dos variables que serveixen per gestionar cóm es gestionarà el sensor.

La primera variable *–distanceFilter-* indica si hi ha una llindar mínim de metres que un usuari hagi de recórrer abans de que cridi el mètode del delegat. Això vol dir que la consulta de la posició GPS es realitzarà i llavors es compararan les posicions antiga i nova de l'usuari i, si és una distancia major al llindar establert per aquesta variable es notificarà al mètode implementat per l'objecte delegat.

La segona variable *–desiredAccuracy-* serveix per dilucidar quin és el mètode d'obtenció de la posició. Hi ha tres possibles mètodes d'obtenció: Satèl·lit, Triangulació d'antenes o via WiFi depenent de si es vol una precisió amb un radi d'error d'un quilòmetre, d'un hectòmetre o d'un decàmetre.

Lògicament, a major precisió més consum de bateria. Aquest fet és força significatiu ja que el consum que realitza una petició de localització per Satèl·lit via GPS és força important i fer un ús intensiu d'aquesta funció redueix dràsticament la durada de la bateria.

Per últim, la darrera crida activa el *CLLocationManager* i un cop hagi obtingut la posició, es produirà una crida asíncrona al mètode del delegat (que es repetirà contínuament cada cop que es doni un canvi de posició) :

```
-(void) locationManager: (CLLocationManager*)manager didUpdateToLocation:
(CLLocation*)newLocation fromLocation: (CLLocation*)oldLocation {
        [loc setLongitude: newLocation.coordinate.logintude];
        [loc setLatitude: newLoctaion.coordinate.latitude];
}
```

C.4. UTILITZANT LA CÀMERA DEL TERMINAL

El seu tractament pel que fa a gestió de la vista és igual al exposat amb la vista de reproducció. Apareix com una vista modal que presenta una informació i implementa 3 botons. El d'enregistrar, el de cancel·lar i el de canviar entre mode instantània i mode de vídeo.

Una altra vegada el patró es repeteix com amb els *frameworks* de localització o d'anotacions: Assignar qui implementa els mètodes del delegat, inicialitzar una instancia del controlador i implementar el mètode corresponent del delegat.

```
UIImagePickerController* picker = [[UIImagePickerController alloc] init];
        picker.delegate = self
       picker.mediaTypes = [UIImagePickerController availableMediaTypesForSourceType:
                            UIImagePickerControllerSourceTypeCamera];
        [self presentModalViewController:picker animated:YES];
        [picker release];
```

Com es veu en aquest codi mínim per a la execució del controlador de media, primerament es crea una instancia del controlador i s'assigna a la variable, en aquest cas *picker*.

De la següent sentencia se'n desprèn que la classe que implementa aquest codi, també implementara els mètodes delegats del *picker* ja que serà ella mateixa qui gestioni els mètodes asíncrons entregats per el controlador.

A continuació cal indicar quin tipus de media serà capaç d'enregistrar el controlador. En aquest cas, ja que es vol obtenir un vídeo de la càmera, s'indica la constant corresponent, *UIImagePickerControllerSourceTypeCamera*.

Per últim, la vista es carrega com ja s'ha dit amb el mètode d'entrada propi de les vistes modals.

Un cop la vista s'ha inicialitzat i és mostrada, la pantalla disposarà la interfície de la figura C.2 a la pàgina següent.

És important comentar que els elements que apareixen a la interfície estan gestionats automàticament per la implementació de la classe, sense que s'hagi de realitzar cap altre tasca addicional (excepte la implementació dels mètodes adoptats del delegat lògicament).

L'usuari pot controlar l'autofocus amb una pulsació sobre la zona on vulgui enfocar la imatge, el mode del flash –automàtic, manual, o desactivat- i la càmera de la que enregistrar, en els models que en disposin de més d'una.

De la botonera inferior ja se n'ha parlat. Els botons de cancel·lar i d'enregistrar estan gestionats per la implementació dels mètodes delegats mentre que el tercer selecciona el tipus de captura que es duu a terme, ja sigui una imatge o un vídeo.

Fig. C.2 Interficie nativa d'iOS per a l'enregistrament de vídeo

Un cop s'ha enregistrat el vídeo, es realitza una crida al mètode implementat pel delegat:
*- (void) imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info ;*

Bàsicament aquest mètode conté un NSDictionary. Aquest NSDictionary és un tipus de classe que pot contenir al seu interior qualsevol tipus d'objecte lligat a una clau, que en aquest cas és una NSString.

D'aquesta manera, s'estableixen relacions de parelles entre claus-objectes de tal forma que si un usuari vol obtenir un objecte, només ha de passar al NSDictionary la clau corresponent.

En el cas de les dades del vídeo, enlloc de proporcionar un vector de bytes contenint tot el vídeo en cru, es proporciona la direcció URL interna on està emmagatzemat. La seva obtenció és la següent:

*NSUrl *url = [info objectForKey:UIImagePickerControllerMediaUrl];*

Com que el vídeo ha d'estar emmagatzemat a un servidor extern, es necessita obtenir el contingut d'arrere aquesta adreça URL en un *array* de bytes per poder enviar-los cap al servidor.

Per convertir la informació que conté una adreça url en bytes s'ha de crear una instancia de la classe NSData amb el constructor apropiat.

*NSData *videoBytes = [[NSData alloc] dataWithContentsOfURL: url];*

Només resta tancar la vista d'enregistrament i enviar el video mitjançant la vista de pujar vídeos ja comentada.